TTK4135

REPORT

Helicopter Lab

April 23, 2015

Student numbers:
712623
742401
742434
742441
742446

Group 9

FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND ELECTRICAL ENGINEERING
DEPARTMENT OF ENGINEERING CYBERNETICS

NTNU – Trondheim
Norwegian University of Science and Technology

Abstract

This is a report for the mandatory lab exercise in the course TTK4135 Optimization and Control held at NTNU the spring 2015. The purpose of this exercise is to get practical experience in formulating dynamic optimization problems, discretise them and solve them using a computer, as well as using these results to implement optimal controllers with and without feedback. The optimization problems that got solved were quadratic minimization problems, and the feedback was introduced using LQ controllers.

Contents

1 Introduction

In this lab called the Helicopter Lab. As the name indicates we are going to control a model of a helicopter in different ways. The system with its hardware is described in detail in [?]. A model of the helicopter with controllers is already made for us. Our objective is to find an optimal sequence of inputs that will steer the helicopter on the desired path. This will be done by formulating optimization problems, and solving them. We will solve them as discrete problems using MATLAB, which mean that we first have to discretise our system. We apply this input sequence to the system using Simulink/QuaRC. We then run the helicopter using the optimal input directly, and compare the behaviour to when we introduce feedback.

In this report we will be presenting our results in the same order that they were obtained, meaning we will have one section for each exercise in [?], where the relevant mathematics, code, plots, results, and so on will be added. The discussion will be done alongside this, with the exception of one summarizing discussion at the end of the report.

2 Repetition/introduction to Simulink/QuaRC (10.1)

The first problem was meant as a repetition for those that have completed the Helicopter lab in the course TTK4115 Linear System Theory, and an introduction to Simulink/QuaRC for everyone else. All members of our group has completed TTK4115, so we didn't use much time on this problem. We ran the Simulink/QuaRC-program, and observed that the pre-made controllers were behaving fine. We changed the setpoints in real-time, and immediately got satisfying responses from the helicopter.

'Smack my pitch up.'

The Prodigy (1997)

3 Optimal control of pitch/travel without feedback (10.2)

3.1 State space form (10.2.1)

We want to write the model (1) in continuous time state space form with states and input as shown in (2) below.

$$\dot{\mathbf{x}} = \boldsymbol{A}_c\mathbf{x} + \boldsymbol{B}_c u \tag{1}$$

$$\begin{aligned} \mathbf{x} &= \begin{bmatrix}\lambda & r & p & \dot{p}\end{bmatrix}^{\mathrm{T}} \\ u &= p_c \end{aligned} \tag{2}$$

The equations of motion for the states are shown in (3) and the constants used are defined in (4). These are derived in detail in [?], and will not be further explained here. This also applies for (20) in section 5.1.

$$\begin{aligned} \dot{\lambda} &= r \\ \dot{r} &= -K_a p \\ \dot{p} &= \dot{p} \\ \ddot{p} &= K_1K_{pp}(p_c - p) - K_1K_{pd}\dot{p} \end{aligned} \tag{3}$$

$$\begin{aligned} K_1 &= \frac{K_f l_n}{J_p} \\ K_2 &= \frac{K_p l_a}{J_t} \end{aligned} \tag{4}$$

The above gives the result in (5).

$$\dot{\mathbf{x}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} \end{bmatrix}}_{\boldsymbol{A}_c}\mathbf{x} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1K_{pp} \end{bmatrix}}_{\boldsymbol{B}_c} u \tag{5}$$

3.2 Model discussion (10.2.1)

The system states are travel, travel rate, pitch, and pitch rate. The controller output is the pitch setpoint to be used by the already implemented controller which in turn calculates voltage inputs for the hardware. We are therefore not modelling the helicopter alone, but a larger system encompassing the helicopter along with the given controller. This corresponds with figure 7 in the exercise text [?].

3.3 Discretisation (10.2.2)

We discretise the system by the Forward Euler Method. The general definition of the method and how it relates to our system is described by equations (6) and (7), respectively.

$$y_{k+1} = y_k + hf(x_k, y_k) \tag{6}$$

$$f = (\boldsymbol{A}_c \mathbf{x}_k + \boldsymbol{B}_c u_k) \tag{7}$$

Using (6) and (7), we can find the matrices of the discrete system, as shown in (8), (9), and (10).

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k + (\boldsymbol{A}_c \mathbf{x}_k + \boldsymbol{B}_c u_k)\, h \\ &= \mathbf{x}_k + h\boldsymbol{A}_c \mathbf{x}_k + h\boldsymbol{B}_c u_k \\ &= (\mathbf{I} + h\boldsymbol{A}_c)\, \mathbf{x}_k + h\boldsymbol{B}_c u_k \\ &= \boldsymbol{A}\mathbf{x}_k + \boldsymbol{B}u_k \end{aligned} \tag{8}$$

$$\boldsymbol{A} = \mathbf{I} + h\boldsymbol{A}_c = \begin{bmatrix} 1 & h & 0 & 0 \\ 0 & 1 & -K_2 h & 0 \\ 0 & 0 & 1 & h \\ 0 & 0 & -K_1 K_{pp} h & 1 - K_1 K_{pd} h \end{bmatrix} \tag{9}$$

$$\boldsymbol{B} = h\boldsymbol{B}_c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1 K_{pp} h \end{bmatrix} \tag{10}$$

3.4 Discussion of the cost function (10.2.3)

We are given the cost function (11) from the exercise text [?] that we want to minimise with the constraints given in (12).

$$\phi = \sum_{i=1}^{N} (\lambda_i - \lambda_f)^2 + q p_{ci}^2, \quad q \geq 0 \tag{11}$$

$$|p_k| \leq \frac{30\pi}{180}, \quad k \in \{1, ..., N\} \tag{12}$$

This cost function focuses on the error between λ_i and λ_f. It is a least-square (i.e. quadratic) function and we will therefore use quadratic programming to minimise the error of λ_i and λ_f. The second part of the cost function consists of the pitch setpoint, which is weighed by a constant q.

3.5 Procedure for solving the optimization problem (10.2.3)

The intention is to calculate an optimal helicopter trajectory between $x_0 = \begin{bmatrix}\lambda_0 & 0 & 0 & 0\end{bmatrix}^{\mathrm{T}}$ and $x_f = \begin{bmatrix}\lambda_f & 0 & 0 & 0\end{bmatrix}^{\mathrm{T}}$, with a constant elevation angle. In this case we use $\lambda_0 = \pi$ and $\lambda_f = 0$.

We start by declaring the constants given in Table 8.2, the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices given in (5), and x_0 given above. Then we implement the constraints in (12) using the following code:

```

% Bounds

ul      = -30*pi/180;         % Lower bound on control -- u1
uu      = 30*pi/180;          % Upper bound on control -- u1

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]        = genBegr2(N,M,xl,xu,ul,uu);
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
```

where `genBegr2` is like this:

```
function [vlb,vub] = genBegr2(N,M,xl,xu,ul,uu);

```

```
vlb_x    = repmat(xl,N,1);    % Lager begrensning for hvert
    ↪ tidsskritt
vub_x    = repmat(xu,N,1);    % Lager begrensning for hvert
    ↪ tidsskritt

vlb_u    = repmat(ul,M,1);    % Lager begrensning for hvert
    ↪ tidsskritt
vub_u    = repmat(uu,M,1);    % Lager begrensning for hvert
    ↪ tidsskritt

vlb      = [vlb_x; vlb_u];    % Setter sammen begrensningene for
    ↪ xl og ul
vub      = [vub_x; vub_u];    % Setter sammen begrensningene for
    ↪ xu og uu
```

After that we find weight matrices for the QP problem and the matrices for the equality constrains:

```
% 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 = q;                        % Weight on input
Q = 2*genq2(Q1,P1,N,M,mu);     % Generate Q
c = zeros(N*mx+M*mu,1);        % Generate c

% Generate system matrixes for linear model

Aeq = gena2(A1,B1,N,mx,mu);    % Generate A
beq = zeros(1, size(Aeq,1));   % Generate b
beq(1:mx) = A1*x0;             % Initial value
```

using `genq2`

```
function Q = genq2(Q1,P1,N,M,mu)

p1  = blkdiag2(P1,M);
q1  = blkdiag2(Q1,N);

Q   = blkdiag(q1,p1);
```

and `gena2`

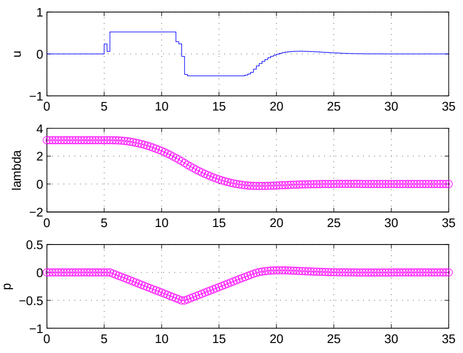
```
function A = gena2(A1,B1,N,mx,mu)

A    = zeros(N*mx,N*mx+N*mu);
b1   = blkdiag2(-B1,N);
a1   = eye(N*mx);
for i=1:(N-1)
  a1([(i*mx+1):((i+1)*mx)],[((i-1)*mx+1):(i*mx)])=-A1;
end
A    = [a1 b1];
```

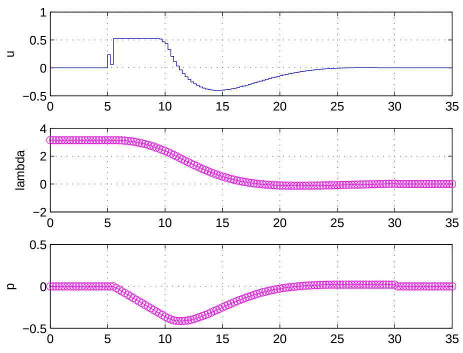
At last we solve the quadratic problem using the MATLAB command `quadprog`:

```
    [z,lambda] = quadprog(Q, c, [], [], Aeq, beq, vlb, vub, z0);
```

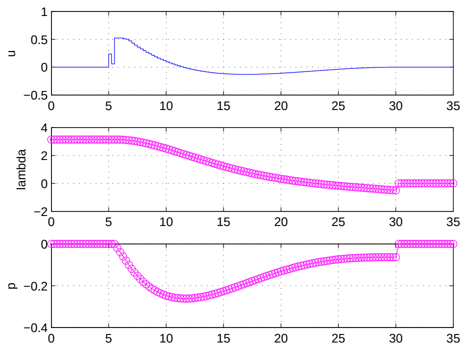
where $\boldsymbol{Z} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_N & u_1 & u_2 & \ldots & u_N \end{bmatrix}^{\mathrm{T}}$. The complete code is listed in the appendix. We extract the desired states and get these plots, using three different values of q:



(a) Optimal path with $q = 0.1$



(b) Optimal path with $q = 1$



(c) Optimal path with $q = 10$

Figure 1: Optimal paths with different q

From the figures we see that a large q will downpress the pitch, resulting in slower travel. This makes sense. (11) tells us that q is the weight of the pitch rate. We want to minimise the entire expression, meaning if we make the weight of p_c higher, p_c has to get smaller to minimize the function.

3.6 Discussion of unwanted effects (10.2.3)

When $\lambda = \lambda_f$, the first part of the cost function will be 0. In this case, the QP algorithm will only attempt to minimise pitch. It achieves this by setting the pitch equal to zero, which most likely would result in overshoot and oscillation.

3.7 Deviation and desired behaviour of helicopter (10.2.4)

Now we want to use the optimal input found in section 3.5 as an input to our helicopter system. We do this using Simulink with QuaRC as seen in figure 2

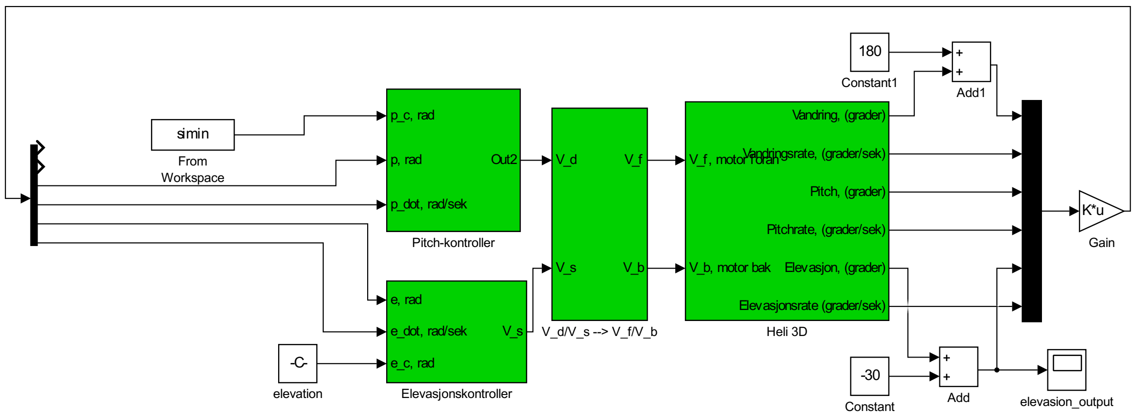


Figure 2: Simulink model of the system

The helicopter does not stop at the desired end point, as the QP algorithm overcompensates for the time needed to brake, and the helicopter starts gliding back in the opposite direction. This is shown in the plots.

Note, however, that we discovered a fault related to the travel sensor causing it to not reset accumulated travel at start-up. As a consequence of this is we got a value of lambda that is off by roughly 6000 degrees, but the general response of the system is still valid and easily seen in the figures.

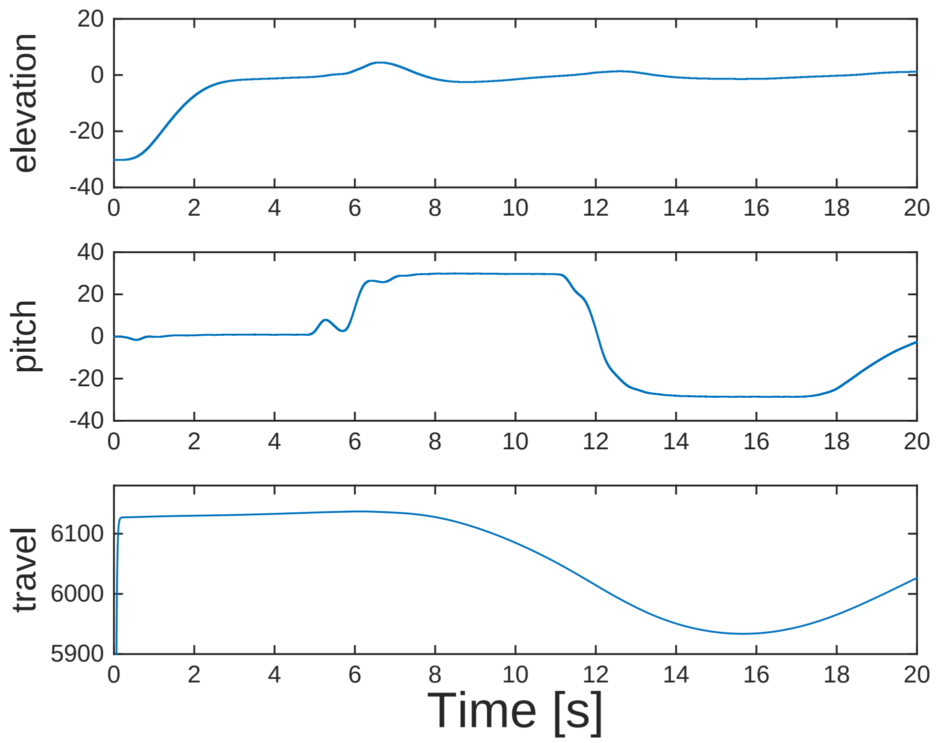


Figure 3: Actual values day 2

If we compare Figure 3 and Figure 1b, we see that the travel values have a similar profile for the first 15 seconds or so, after which it starts to overcompensate (as described earlier). There is some deviation of pitch, with the actual pitch first swaying one way, and then back the other. We can also see that the elevation is not constant, as opposed to what we assumed.

'I've got 99 problems, but a pitch ain't one.'

Ice-T (1993)

4 Optimal control of pitch/travel with LQ control (10.3)

4.1 LQ controller (10.3.1)

We introduce feedback to our system using the following manipulated variable:

$$\mathbf{u}_k = \mathbf{u}_k^* - \boldsymbol{K}^{\mathrm{T}}(\mathbf{x}_k - \mathbf{x}_k^*) \tag{13}$$

where u_k^* and x_k^* are the optimal input sequence and optimal trajectory calculated in the previous task. $\boldsymbol{K}$ is the *gain matrix*, and can be calculated in many ways. We are going to determine it with an LQ (linear quadratic) controller that minimises the function

$$J = \sum_{i=0}^{\infty} \Delta\mathbf{x}_{i+1}^{\mathrm{T}}\boldsymbol{Q}\Delta\mathbf{x}_{i+1} + \Delta\mathbf{u}_i^{\mathrm{T}}\boldsymbol{R}\Delta\mathbf{u}_i, \quad \boldsymbol{Q} \geq 0,\, \boldsymbol{R} > 0 \tag{14}$$

for the linear model

$$\Delta\mathbf{x}_{i+1} = \boldsymbol{A}\Delta\mathbf{x}_i + \boldsymbol{B}\Delta\mathbf{u}_i \tag{15}$$

where $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}^*$ and $\Delta\mathbf{u} = \mathbf{u} - \mathbf{u}^*$ are the deviations from the optimal trajectory. (14) can be solved in MATLAB using the function `dlqr`.

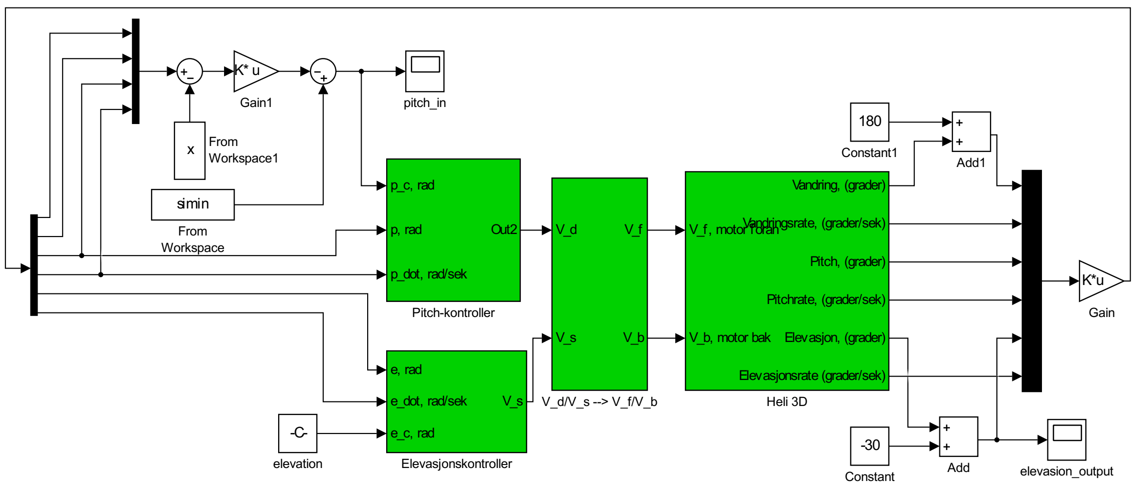


Figure 4: Simulink model of the system

The behaviour of the helicopter is tuned by altering $\boldsymbol{Q}$ and R, and by trial and error we obtained the following values: (TODO: HOW GOOD WAS THIS?)

$$\boldsymbol{Q} = \begin{bmatrix} 25 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix} \tag{16}$$

$$R = 1 \tag{17}$$

4.2 Behaviour with feedback (10.3.2)

The first thing we notice is that the helicopter now does come to a halt at the setpoint. In the plots of the measured states, you can see that the travel is stationary from 19 seconds onwards. You can also see that it travels a bit past its reference λ_f, but that it goes back before it stops its travel.

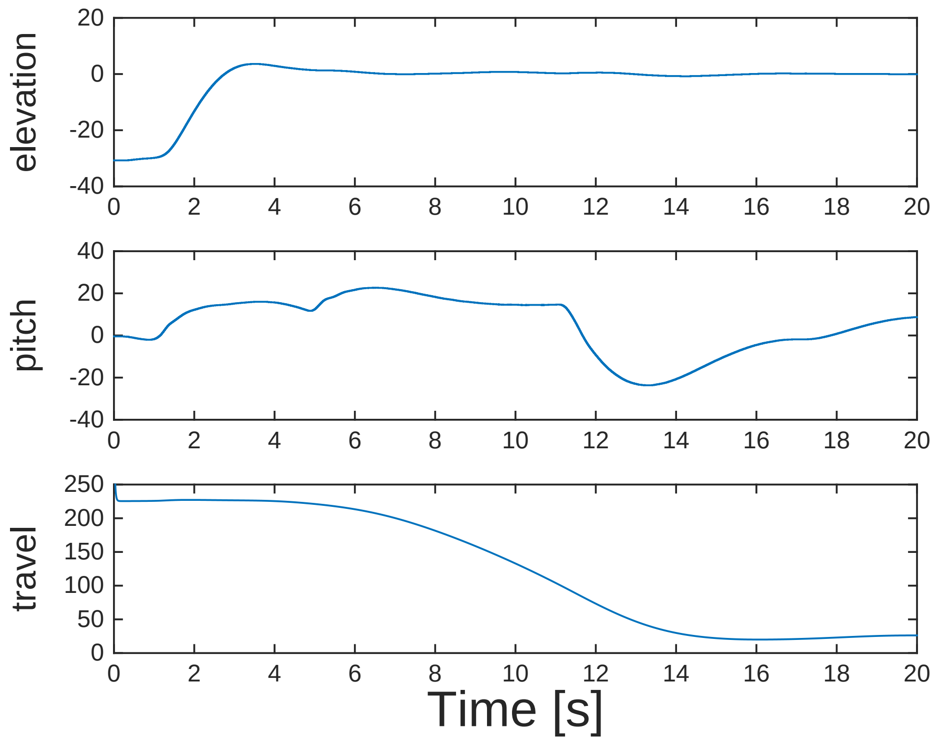


Figure 5: Actual values day 3

As shown the shape of both the travel and pitch match the calculated values found in **??** well. The feedback distorts the pitch from the calculated optimal as it works in parallel with the optimal input for the helikopter.

4.3 MPC discussion (10.3.3)

4.3.1 Realization of MPC (10.3.3)

An MPC can be realized with the following algorithm for linear MPC with state feedback

```
for t = 0,1,2,... do
    Get the current state $x_t$
    Solve the convex QP problem on the prediction horizon from $t$ to $t+N$ with $x_t$ as the initial condition
    Apply the first control move $u_t$ from the solution above
end for
```

4.3.2 Advantages and disadvantages with using MPC (10.3.3)

There are both advantages and disadvantages with using an MPC controller compared to an LQR controller. MPC requires a lot of computations, and can therefore require a lot of computer resources. All these computations also take quite a bit of time, and is slower than an LQR controller. However, all of these computations mean that you get a very accurate controller. With the LQR you won't get that many computations, so it will be faster and require less computational power, but it will also be less accurate.

4.3.3 Figure 8 with MPC (10.3.3)

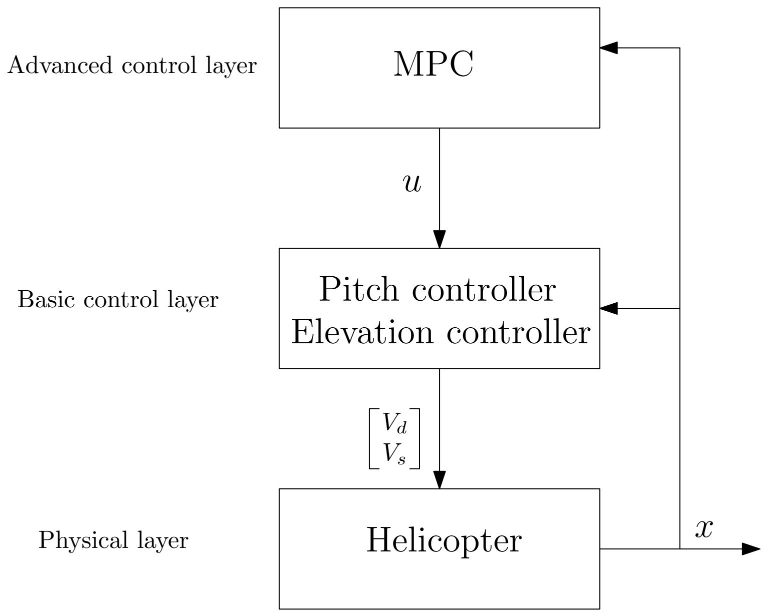


Figure 6: Figure 8 with MPC

The MPC, pitch controller and elevation controller gets the actual position x from the helicopter and calculates the input u, which the pitch and elevation controllers then use to find the voltages V_d and V_s to control the motors

'A helicopter does not want to fly, it just vibrates so much that the ground rejects it.'

Kurt Vonnegut

5 Optimal control of pitch/travel and elevation with and without feedback (10.4)

5.1 Discretisation with elevation states (10.4.1)

We will re-formulate the continuous system (18) with additional states for elevation (e) and elevation rate ($\dot{e}$). The system with new state and input vectors (19) and state equations (20) yield the matrices shown in (21).

$$\dot{\mathbf{x}} = \boldsymbol{A}\mathbf{x} + \boldsymbol{B}\mathbf{u} \tag{18}$$

$$\mathbf{x} = \begin{bmatrix}\lambda & r & p & \dot{p} & e & \dot{e}\end{bmatrix}^{\mathrm{T}}, \qquad \mathbf{u} = \begin{bmatrix}p_c & e_c\end{bmatrix}^{\mathrm{T}} \tag{19}$$

$$\begin{aligned}
\dot{\lambda} &= r \\
\dot{r} &= -K_2 p \\
\dot{p} &= \dot{p} \\
\ddot{p} &= K_1 K_{pp}(p_c - p) - K_1 K_{pd}\dot{p} \\
\dot{e} &= \dot{e} \\
\ddot{e} &= K_3 K_{ep}(e_c - e) - K_3 K_{ed}\dot{e}
\end{aligned} \tag{20}$$

$$\dot{\mathbf{x}} = \underbrace{\begin{bmatrix}0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3K_{ep} & -K_3K_{ed}\end{bmatrix}}_{\boldsymbol{A}_c} \mathbf{x} + \underbrace{\begin{bmatrix}0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3K_{ep}\end{bmatrix}}_{\boldsymbol{B}_c} \mathbf{u} \tag{21}$$

5.2 Discretisation again (10.4.2)

As in section 3.3, we again need to discretize the system. The result can be seen in (23) and (24).

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k + (\boldsymbol{A}_c\mathbf{x}_k + \boldsymbol{B}_c\mathbf{u}_k)\, h \\ &= \underbrace{(\mathbf{I} + h\boldsymbol{A}_c)}_{\boldsymbol{A}}\mathbf{x}_k + \underbrace{h\boldsymbol{B}_c}_{\boldsymbol{B}}\mathbf{u}_k \end{aligned} \tag{22}$$

$$\boldsymbol{A} = \begin{bmatrix} 1 & h & 0 & 0 & 0 & 0 \\ 0 & 1 & -hK_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & h & 0 & 0 \\ 0 & 0 & -hK_1K_{pp} & 1 - hK_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & h \\ 0 & 0 & 0 & 0 & -hK_3K_{ep} & 1 - hK_3K_{ed} \end{bmatrix} \tag{23}$$

$$\boldsymbol{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ hK_1K_{pp} & 0 \\ 0 & 0 \\ 0 & hK_3K_{ep} \end{bmatrix} \tag{24}$$

5.3 Solution of the optimization problem (10.4.3)

The cost function now has to include the elevation state, which gives us (25), with constrains given by (26).

$$\phi = \sum_{i=1}^{N} (\lambda_i - \lambda_f)^2 + q_1 p_{ci}^2 + q2e_{ci}^2 \tag{25}$$

$$\alpha \exp(-\beta(\lambda_k - \lambda_t)^2) - e_k \leq 0, \quad k \in \{1, ..., N\} \tag{26}$$

In (26), we used the values $\alpha = 0.2$, $\beta = 20$ and $\lambda_t = \frac{2\pi}{3}$. We use `fmincon` to get the result of the cost function with the constraints. The parameters we send it are `vlb` and `vub`, which are vectors containing the linear constraints, `mycon`, which returns the nonlinear constraints and `Aeq` and `beq` which are the matrices for the problem to be solved. We also send in the cost function itself and the start values `z0`.

```
costf = @(z) 0.5*z'*Q*z;
z = fmincon(costf, z0, [], [], Aeq, beq, vlb, vub, @mycon);
```

5.4 Comparison of performance with and without feedback(10.4.4)

Without feedback the pitch goes to 0 when $\lambda = \lambda_f$, but does not compensate to stop the helicopter from traveling past $\lambda = \lambda_f$. With feedback, the helicopter will change its pitch to the opposite direction, making the helicopter stop and stay still at $\lambda = \lambda_f$.

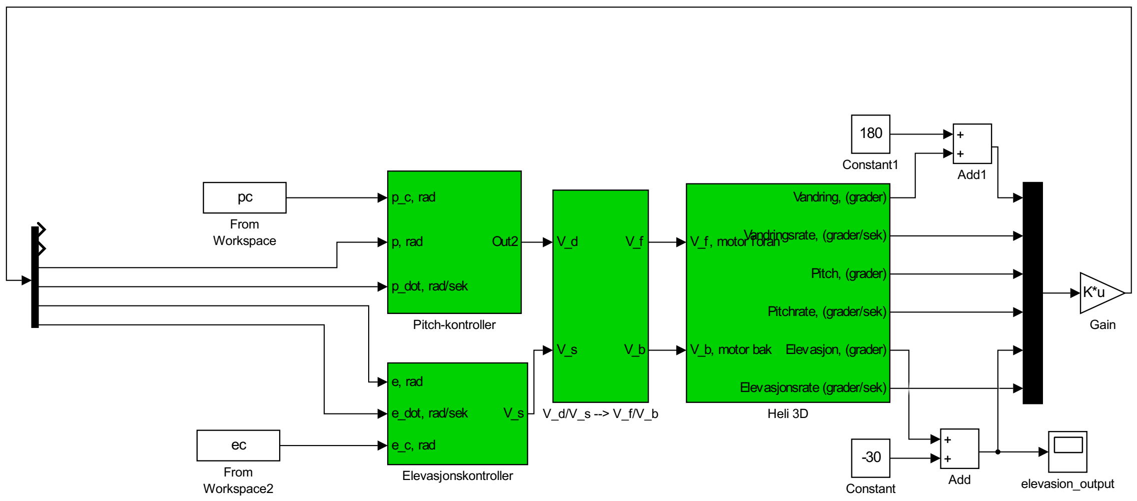


Figure 7: Simulink model of the system without feedback

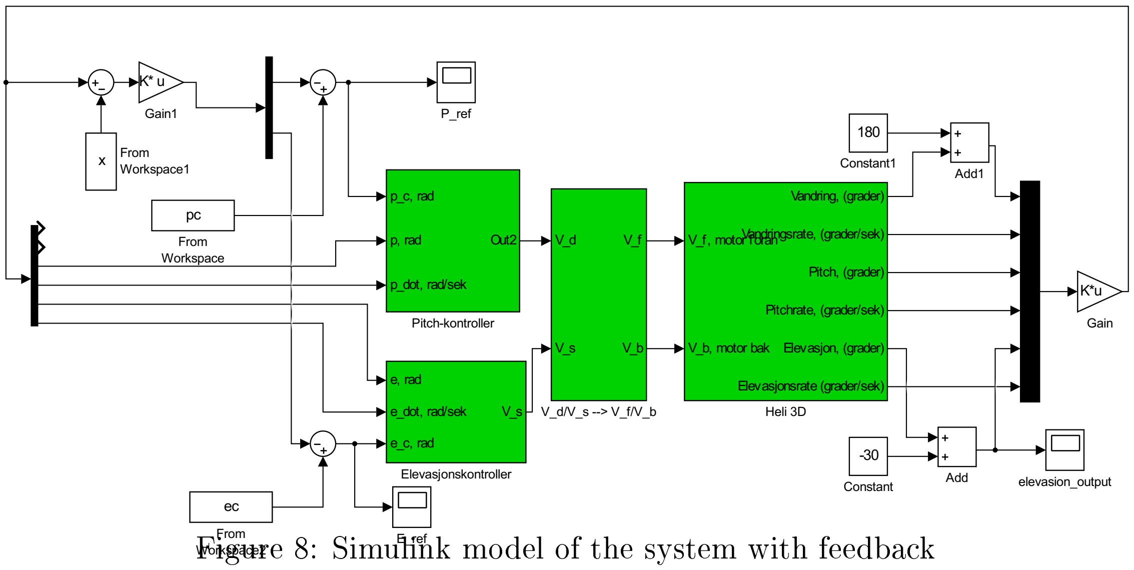


Figure 8: Simulink model of the system with feedback

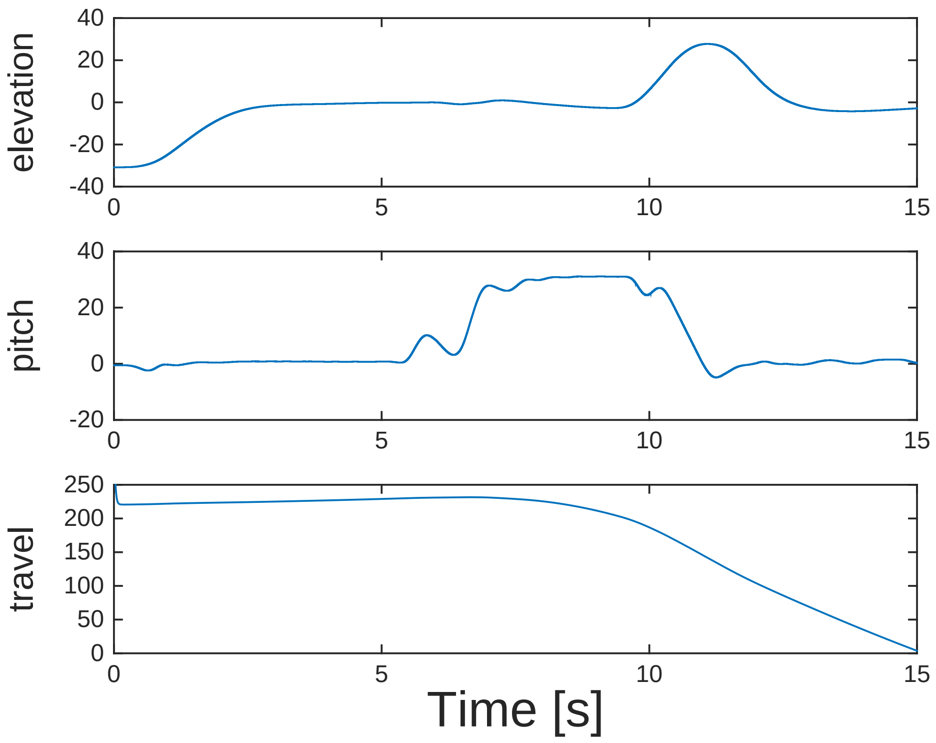


Figure 9: Actual values day 4 without feedback

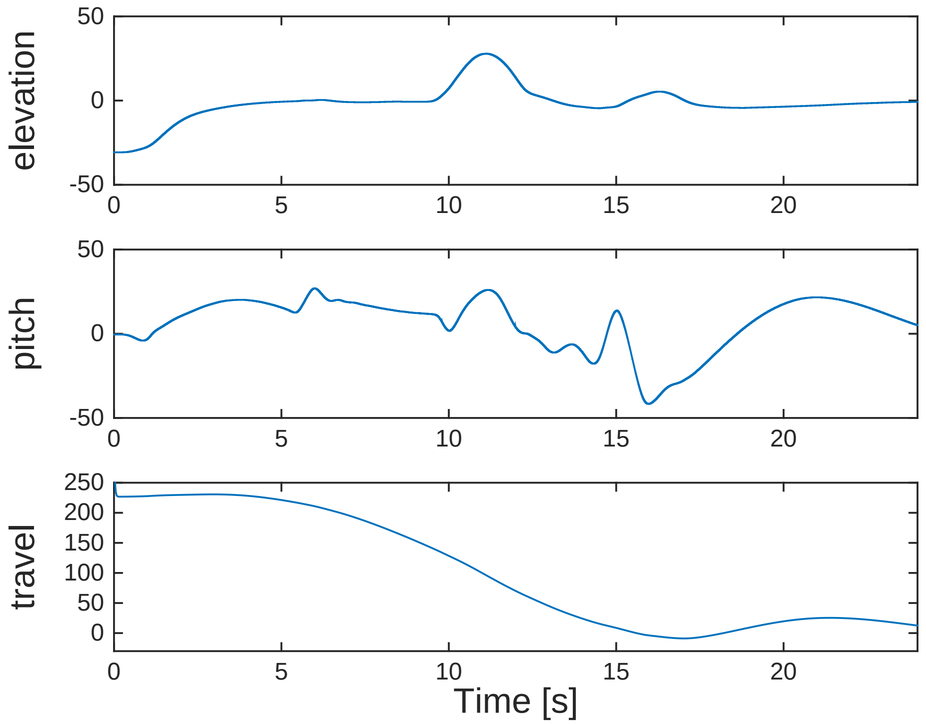


Figure 10: Actual values day 4 with feedback

As we can see in the sensor data from the model without feedback, the helicopter does not stop at $\lambda = \lambda_f$, but keeps gliding. In the model with feedback, however, it does stop at $\lambda = \lambda_f$, but the change in pitch to keep it still in the reference point also makes the helicopter elevate a bit, as can be seen in the plot at around 16 seconds. It also goes a bit past the reference, but goes back to the reference and stays there.

In the calculated outputs you can see that it also expects some change in the pitch when trying to stop at the reference point, but it is much more subtle than what happens, especially with feedback. This would be closer if the LQR was tuned better. In the sensor values of the model without feedback, this is more similar to the calculated pitch, but the travel is a bit off, as the helicopter keeps gliding after it reaches the reference point. This is better in the model with feedback, as it does stop in the reference point, even though it does not stop as fast and smooth as the calculated output would suggest.

5.5 Discussion of decoupled states (10.4.5)

In real life the pitch and pitch rate of the helicopter would have an effect on both the elevation rate and the elevation itself. This will make the helicopter get an offset on where it goes compared to the calculated optimal trajectory. By modelling the system with the elevation and travel coupled with the pitch, you will remove the offset at the cost of a more complex model.

6 Discussion

The purpose of this lab was to formulate a dynamic optimization problem, and solve this using `MATLAB` and Simulink, as well as seeing the results of control both with and without feedback.

First, this was done by using model-based optimization to provide the pitch and elevation controllers with an input u, before the model was improved by adding an LQR-control layer between the optimization layer and the basic control layer.

7 Conclusion

In conclusion our group feels that it has been interesting to work on this lab, and that we have learnt a lot by doing it. By getting a practical approach to implementing optimization algorithms, we have gained insight into how it can be to work as a control system engineer and some of the problems they encounter. We have also seen that using optimization algorithms when making a control system makes the system more precise, and is worth both the time and energy it takes to implement them.

All in all we think this has been a successful exercise, that has been both rewarding and educational.

8 Appendix

8.1 Variables

Symbol	Variable
p	Pitch
p_c	Setpoint for pitch
λ	Travel
r	Speed of travel
r_c	Setpoint for speed of travel
e	Elevation
e_c	Setpoint for elevation
V_f	voltage, motor in front
V_b	Voltage, motor in back
V_d	Voltage difference, $V_f - V_b$
V_s	Voltage sum, $V_f + V_b$
K_{pp}, K_{pd}, K_{ep}, K_{ei}, K_{ed}	Controller gains
T_g	Moment needed to keep the helicopter flying

8.2 Constants

Symbol	Parameter	Value	Unit
l_a	Distance from elevation axis to helicopter body	0.64	m
l_h	Distance from pitch axis to motor	0.177	m
K_f	Distance from elevation axis to helicopter body	0.1983	$\mathrm{N\,m^{-1}}$
J_e	Moment of inertia for elevation	1.0625	$\mathrm{kg\,m^2}$
J_t	Moment of inertia for travel	1.0625	$\mathrm{kg\,m^2}$
J_p	Moment of inertia for pitch	0.0406	$\mathrm{kg\,m^2}$
m_h	Mass of helicopter	1.297	kg
m_w	Balance weight	1.802	kg
m_g	Effective mass of the helicopter	0.026	kg
K_p	Force to lift the helicopter from the ground	0.2551	N

8.3 Code for (10.2) and (10.3)

This code is a modified version of the code given out on it's learning.

```

init04;
delta_t   = 0.25;                       % sampling time
sek_forst = 5;

q = 0.1;
% System model. x=[lambda r p p_dot]'

A1 = [1 delta_t 0 0;
      0 1 -K_2*delta_t 0;
      0 0 1 delta_t;
      0 0 -K_1*K_pp*delta_t 1-K_1*K_pd*delta_t];

B1 = [0; 0; 0; K_1*K_pp*delta_t];

% Number of states and inputs

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

% Initial values

x1_0 = pi;                              % Lambda
x2_0 = 0;                               % r
x3_0 = 0;                               % p
x4_0 = 0;                               % p_dot
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 -- u1
uu       = 30*pi/180;                   % Upper bound on control -- u1

```

```
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]       = genBegr2(N,M,xl,xu,ul,uu);
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) = q;                              % Weight on state x3
Q1(4,4) = 0;                              % Weight on state x4
P1 = q;                                   % Weight on input
Q = 2*genq2(Q1,P1,N,M,mu);                % Generate Q
c = zeros(N*mx+M*mu,1);                   % Generate c

% Generate system matrixes for linear model

Aeq = gena2(A1,B1,N,mx,mu);            % Generate A
beq = zeros(1, size(Aeq,1));           % Generate b
beq(1:mx) = A1*x0;                        % Initial value

% Solve Qp problem with linear model
tic
[z,lambda] = quadprog(Q, c, [], [], Aeq, beq, vlb, vub, z0);
t1=toc;


% 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

Antall = 5/delta_t;
Nuller = zeros(Antall,1);
Enere  = ones(Antall,1);

u    = [Nuller; u; Nuller];
x1   = [pi*Enere; x1; Nuller];
x2   = [Nuller; x2; Nuller];
x3   = [Nuller; x3; Nuller];
x4   = [Nuller; x4; Nuller];

%save trajektor1ny
t = 0:delta_t:delta_t*(length(u)-1); % real time

simin = [t' u];

% figure


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')

x = [t' x1 x2 x3 x4];
%%

Q_k = [25 0 0 0;
       0 0.5 0 0;
       0 0 100 0;
       0 0 0 0.5];
R = 1;

[K S E] = dlqr(A1, B1, Q_k, R, 0);
```

8.4 Code for (10.4)

This code is a modified version of the code given out on it's learning.

```matlab

init04;
delta_t    = 0.25;                        % sampling time
sek_forst = 5;

q = 0.1;
% System model. x=[lambda r p p_dot]'

A = [1 delta_t 0 0 0 0;
      0 1 -K_2*delta_t 0 0 0;
      0 0 1 delta_t 0 0;
      0 0 -K_1*K_pp*delta_t 1-K_1*K_pd*delta_t 0 0;
      0 0 0 0 1 delta_t;
      0 0 0 0 -delta_t*K_3*K_ep 1-delta_t*K_3*K_ed];

B = [0 0; 0 0; 0 0; K_1*K_pp*delta_t 0; 0 0; 0 delta_t*K_3*K_ep ];

% Number of states and inputs

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

% Initial values

x1_0 = pi;                                % Lambda
x2_0 = 0;                                 % r
x3_0 = 0;                                 % p
x4_0 = 0;                                 % p_dot
x5_0 = 0;                                 % e
x6_0 = 0;                                 % e_dot
x0   = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0]';          % Initial values

u1_0 = 0;
u2_0 = 0;

q_1 = 1;
q_2 = 1;
alfa = 0.2;
beta = 20;
lambda_t = 2*pi/3;
```

```

% Time horizon and initialization

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


% Bounds

ul       = -30*pi/180;                  % Lower bound on control
   ↪  -- u1
uu       = 30*pi/180;                   % Upper bound on control
   ↪  -- u1

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]       = genBegr2(N,M,xl,xu,ul,uu);
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 system matrixes for linear model
Aeq = gena2(A,B,N,mx,mu);          % Generate A
beq = zeros(1, size(Aeq,1));         % Generate b
beq(1:mx) = A*x0;                       % Initial value

% 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) = q;                            % Weight on state x3
```

```
Q1(4,4) = 0;                              % Weight on state x4
Q1(5,5) = 0;
Q1(6,6) = 0;
P1 = zeros(mu, mu);
P1(1,1) = q_1;                              % Weight on input
P1(2,2) = q_2;
Q = 2*genq2(Q1,P1,N,M,mu);                % Generate Q
c = zeros(N*mx+M*mu,1);                   % Generate c

% Solve Qp problem with linear model

costf = @(z) 0.5*z'*Q*z;
%tic
%[z,lambda] = quadprog(Q, c, [], [], Aeq, beq, vlb, vub, z0);
%t1=toc;
nonlcon = @(z) alfa*exp(-beta*(z(1+mx)-lambda_t)^2)-z(5+mx);
z = fmincon(costf, z0, [], [], Aeq, beq, vlb, vub, @mycon);
% 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

u1 = [u1_0; z(N*mx+1:mu:N*mx+M*mu)]; % Control input from
    ↪ solution
u2 = [u2_0; z(N*mx+2:mu:N*mx+M*mu)];

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
x5 = [x0(5);z(5:mx:N*mx)];
x6 = [x0(6);z(6:mx:N*mx)];

Antall = 5/delta_t;
Nuller = zeros(Antall,1);
Enere  = ones(Antall,1);

u1   = [Nuller; u1; Nuller];
u2   = [Nuller; u2; Nuller];

x1  = [x1_0*Enere; x1; Nuller];
x2  = [Nuller; x2; Nuller];
x3  = [Nuller; x3; Nuller];
```

```
x4  = [Nuller; x4; Nuller];
x5 = [Nuller; x5; Nuller];
x6 = [Nuller; x6; Nuller];

%save trajektor1ny
t = 0:delta_t:delta_t*(length(u1)-1); % real time

pc = [t' u1];
ec = [t' u2];
% figure


figure(2)
subplot(8,1,1)
stairs(t,u1),grid
ylabel('u1')
subplot(8,1,2)
stairs(t, u2),grid
ylabel('u2')
subplot(8,1,3)
plot(t,x1,'m',t,x1,'mo'),grid
ylabel('lambda')
subplot(8,1,4)
plot(t,x2,'m',t,x2','mo'),grid
ylabel('r')
subplot(8,1,5)
plot(t,x3,'m',t,x3,'mo'),grid
ylabel('p')
subplot(8,1,6)
plot(t,x4,'m',t,x4','mo'),grid
ylabel('pdot')
subplot(8,1,7)
plot(t,x5,'m',t,x5','mo'),grid
ylabel('e')
subplot(8,1,8)
plot(t,x6,'m',t,x6','mo'),grid
ylabel('edot')
xlabel('tid (s)')

x = [t' x1 x2 x3 x4 x5 x6];

%%
Q_k = [2 0 0 0 0 0;
       0 1 0 0 0 0;
       0 0 1 0 0 0;
       0 0 0 1.2 0 0;
       0 0 0 0 1 0;
       0 0 0 0 0 1.5];

```

```
R = [1 0;
     0 1];

[K S E] = dlqr(A, B, Q_k, R, 0);
```