

TTK4135

LAB REPORT

# Real-world application of optimal control methods

April 27, 2015

**Group 9**

***Participants:***

712623, 742401, 742434, 742441, 742446

## **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.

FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND  
ELECTRICAL ENGINEERING

DEPARTMENT OF ENGINEERING CYBERNETICS



**NTNU – Trondheim**  
Norwegian University of  
Science and Technology

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Repetition/introduction to Simulink/QuaRC (10.1)</b>	<b>3</b>
<b>3</b>	<b>Optimal control of pitch/travel without feedback (10.2)</b>	<b>3</b>
3.1	State space form (10.2.1) . . . . .	3
3.2	Model discussion (10.2.1) . . . . .	4
3.3	Discretisation (10.2.2) . . . . .	4
3.4	Discussion of the cost function (10.2.3) . . . . .	5
3.5	Procedure for solving the optimization problem (10.2.3) . . . . .	5
3.6	Discussion of unwanted effects (10.2.3) . . . . .	7
3.7	Deviation and desired behaviour of helicopter (10.2.4) . . . . .	7
<b>4</b>	<b>Optimal control of pitch/travel with LQ control (10.3)</b>	<b>10</b>
4.1	LQ controller (10.3.1) . . . . .	10
4.2	Behaviour with feedback (10.3.2) . . . . .	11
4.3	MPC discussion (10.3.3) . . . . .	12
4.3.1	Realization of MPC . . . . .	12
4.3.2	Advantages and disadvantages of using MPC) . . . . .	12
4.3.3	Figure 8 with MPC . . . . .	13
<b>5</b>	<b>Optimal control of pitch/travel and elevation with and without feedback (10.4)</b>	<b>13</b>
5.1	Discretisation with elevation states (10.4.1) . . . . .	13
5.2	Discretisation again (10.4.2) . . . . .	14
5.3	Solution of the optimization problem (10.4.3) . . . . .	15
5.4	Comparison of performance with and without feedback(10.4.4) . . . . .	15
5.5	Discussion of decoupled states (10.4.5) . . . . .	20
<b>6</b>	<b>Discussion</b>	<b>20</b>
<b>7</b>	<b>Conclusion</b>	<b>20</b>
<b>8</b>	<b>Appendix</b>	<b>21</b>
8.1	Variables . . . . .	21
8.2	Constants . . . . .	21
8.3	Code for (10.2) and (10.3) . . . . .	21
8.4	Code for (10.4) . . . . .	24

# 1 Introduction

The intention of this lab assignment is to apply various optimisation-based control techniques to a physical model of a helicopter. The system with its hardware is described in detail in [?]. A mathematical model of the helicopter with pitch and elevation 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.

## 2 Repetition/introduction to Simulink/QuaRC (10.1)

The first problem was intended as 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 have 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.

$$\dot{\mathbf{x}} = \mathbf{A}_c \mathbf{x} + \mathbf{B}_c u \quad (1)$$

$$\begin{aligned} \mathbf{x} &= [\lambda \quad r \quad p \quad \dot{p}]^T \\ u &= p_c \end{aligned} \quad (2)$$

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. The same applies for (20) in section 5.1.) A complete table of symbols and constants lies in the appendix.

$$\begin{aligned} \dot{\lambda} &= r \\ \dot{r} &= -K_a p \\ \dot{p} &= \dot{p} \\ \ddot{p} &= K_1 K_{pp} (p_c - p) - K_1 K_{pd} \dot{p} \end{aligned} \quad (3)$$

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

The above gives the result in (5).

$$\dot{\mathbf{x}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} \end{bmatrix}}_{\mathbf{A}_c} \mathbf{x} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1 K_{pp} \end{bmatrix}}_{\mathbf{B}_c} u \tag{5}$$

### 3.2 Model discussion (10.2.1)

The system states are travel, travel rate, pitch, and pitch rate. The output of our controller 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 consisting of 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.

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

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

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

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

$$\mathbf{A} = \mathbf{I} + h\mathbf{A}_c = \begin{bmatrix} 1 & h & 0 & 0 \\ 0 & 1 & -K_2h & 0 \\ 0 & 0 & 1 & h \\ 0 & 0 & -K_1K_{pp}h & 1 - K_1K_{pd}h \end{bmatrix} \quad (9)$$

$$\mathbf{B} = h\mathbf{B}_c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1K_{pp}h \end{bmatrix} \quad (10)$$

### 3.4 Discussion of the cost function (10.2.3)

The assignment text [?] specifies a cost function (11) that we want to minimise subject to the constraints given in (12).

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

$$|p_k| \leq \frac{30\pi}{180}, \quad k \in \{1, \dots, N\} \quad (12)$$

This cost function focuses on the error between  $\lambda_i$  and  $\lambda_f$ . It is a least-square (i.e. quadratic) function and can therefore be solved by quadratic programming methods. The second term of the cost function involves 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  $x_0 = [\lambda_0 \ 0 \ 0 \ 0]^T$  and  $x_f = [\lambda_f \ 0 \ 0 \ 0]^T$ , while assuming a constant elevation angle. In this case we use  $\lambda_0 = \pi$  and  $\lambda_f = 0$ .

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

```

1 % Bounds
2
3 ul      = -30*pi/180;          % Lower bound on control -- ul
4 uu      = 30*pi/180;          % Upper bound on control -- ul
5
6 xl      = -Inf*ones(mx,1);    % Lower bound on states (no bound)
7 xu      = Inf*ones(mx,1);     % Upper bound on states (no bound)

```

```

8 x1(3) = ul; % Lower bound on state x3
9 xu(3) = uu; % Upper bound on state x3
10
11 % Generate constraints on measurements and inputs
12
13 [v1b,vub] = genBegr2(N,M,x1,xu,ul,uu);
14 v1b(N*mx+M*mu) = 0; % We want the last input to be zero
15 vub(N*mx+M*mu) = 0; % We want the last input to be zero

```

where `genBegr2` was given on it's learning.

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

```

1 % Generate the matrix Q and the vector c (objecitive function
  ↳ weights in the QP problem)
2
3 Q1 = zeros(mx,mx);
4 Q1(1,1) = 1; % Weight on state x1
5 Q1(2,2) = 0; % Weight on state x2
6 Q1(3,3) = 0; % Weight on state x3
7 Q1(4,4) = 0; % Weight on state x4
8 P1 = q; % Weight on input
9 Q = 2*genq2(Q1,P1,N,M,mu); % Generate Q
10 c = zeros(N*mx+M*mu,1); % Generate c
11
12 % Generate system matrixes for linear model
13
14 Aeq = gena2(A1,B1,N,mx,mu); % Generate A
15 beq = zeros(1, size(Aeq,1)); % Generate b
16 beq(1:mx) = A1*x0; % Initial value

```

using `genq2` and `gena2` given on it's learning.

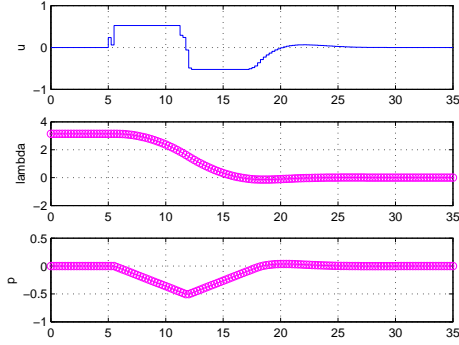
At last we solve the quadratic problem using the MATLAB command `quadprog`:

```

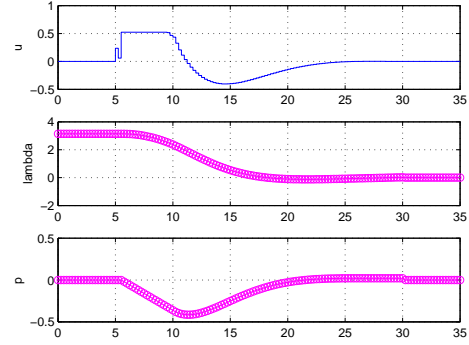
1 [z,lambda] = quadprog(Q, c, [], [], Aeq, beq, v1b, vub, z0);

```

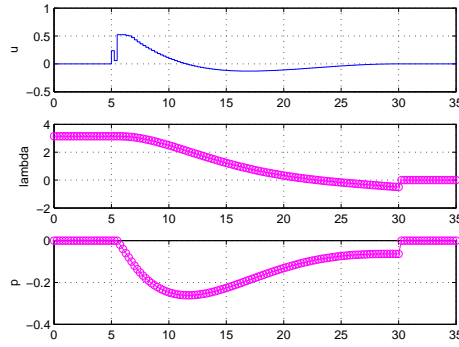
where  $\mathbf{Z} = [\mathbf{x}_1 \ \mathbf{x}_2 \ \dots \ \mathbf{x}_N \ u_1 \ u_2 \ \dots \ u_N]^T$ . The full code is listed in the appendix. We extract the desired states and get the following plots, using three different values of  $q$ :



(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 increase the weight of  $p_c$ ,  $p_c$  will decrease, 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 may result in overshoot and oscillations.

### 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



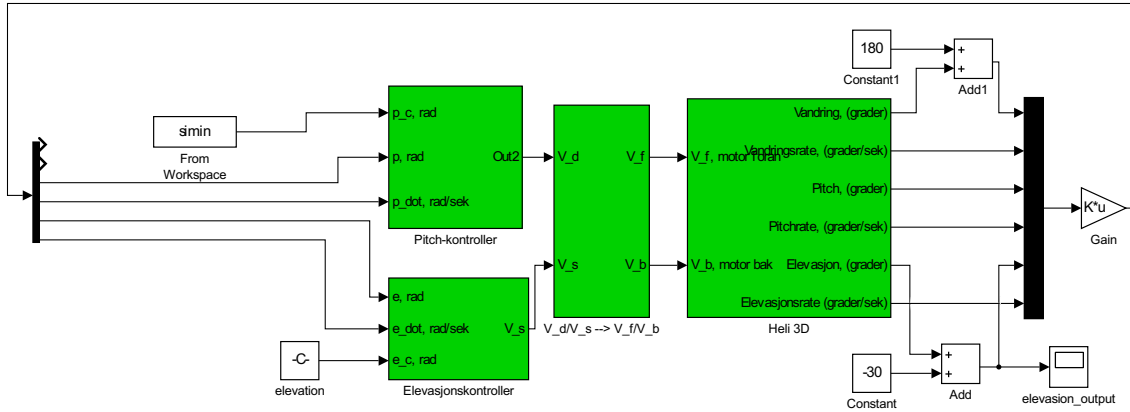


Figure 2: Simulink model of the system

The helicopter does not stop at the desired end point, as the QP algorithm overcompensates for the time needed to brake, and the helicopter glides back in the opposite direction. This can be seen 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 is easily seen in the figures.

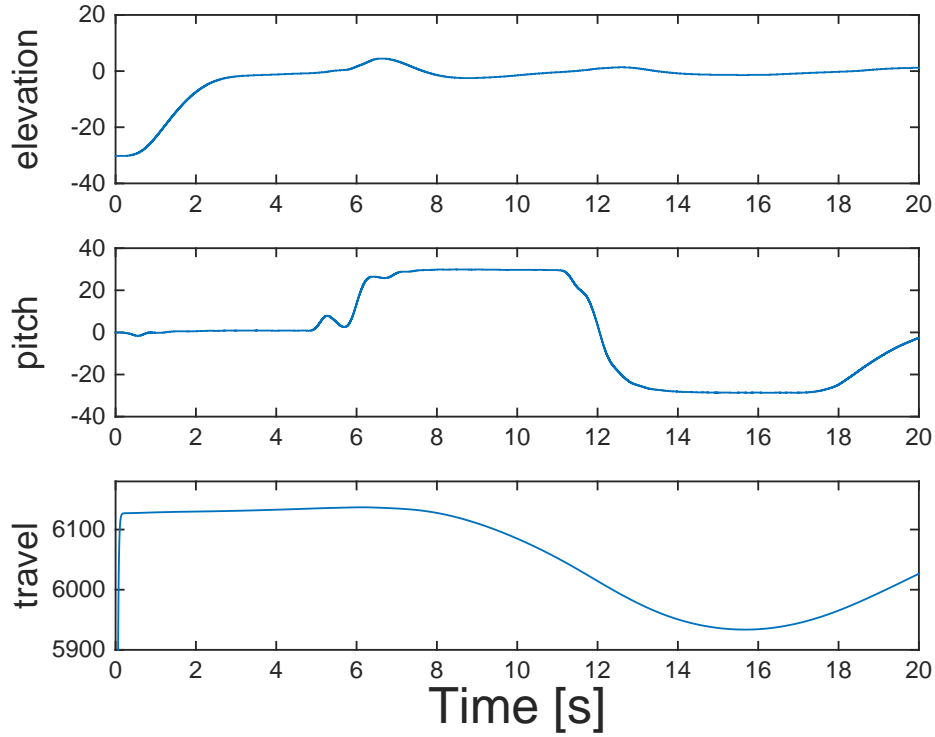


Figure 3: Running the helicopter with optimal input

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 our earlier assumption.

*'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:

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

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

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

for the linear model

$$\Delta \mathbf{x}_{i+1} = \mathbf{A} \Delta \mathbf{x}_i + \mathbf{B} \Delta \mathbf{u}_i \quad (15)$$

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`.

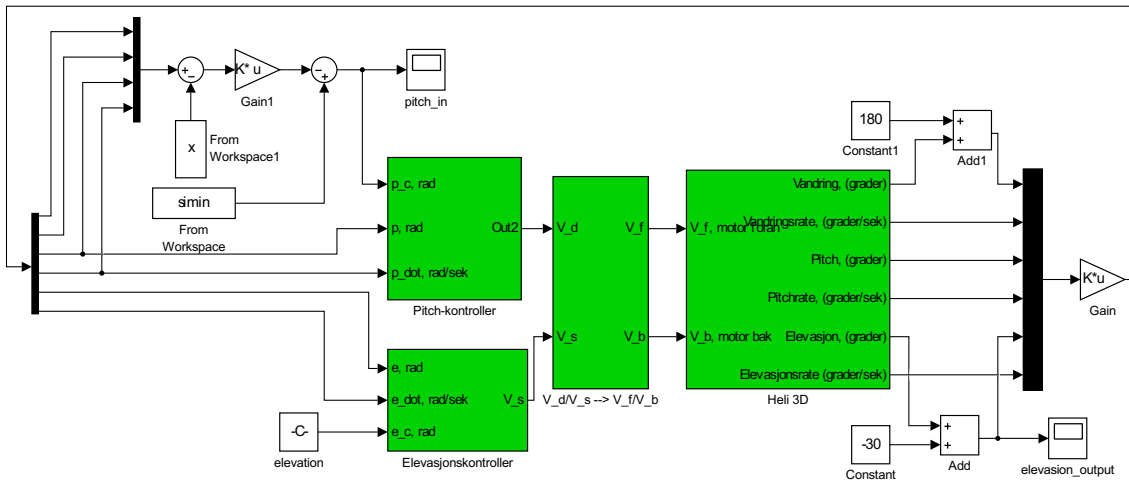


Figure 4: Simulink model of the system

The behaviour of the helicopter is tuned by altering  $\mathbf{Q}$  and  $R$ , and by trial and error we obtained values that yielded descent behaviour of the helicopter:

$$\mathbf{Q} = \begin{bmatrix} 25 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix} \quad (16)$$

$$R = 1 \quad (17)$$

## 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 value is stationary from 19 seconds onwards. You can also see that travel overshoots very slightly, but stabilises nicely at  $\lambda_f$ .

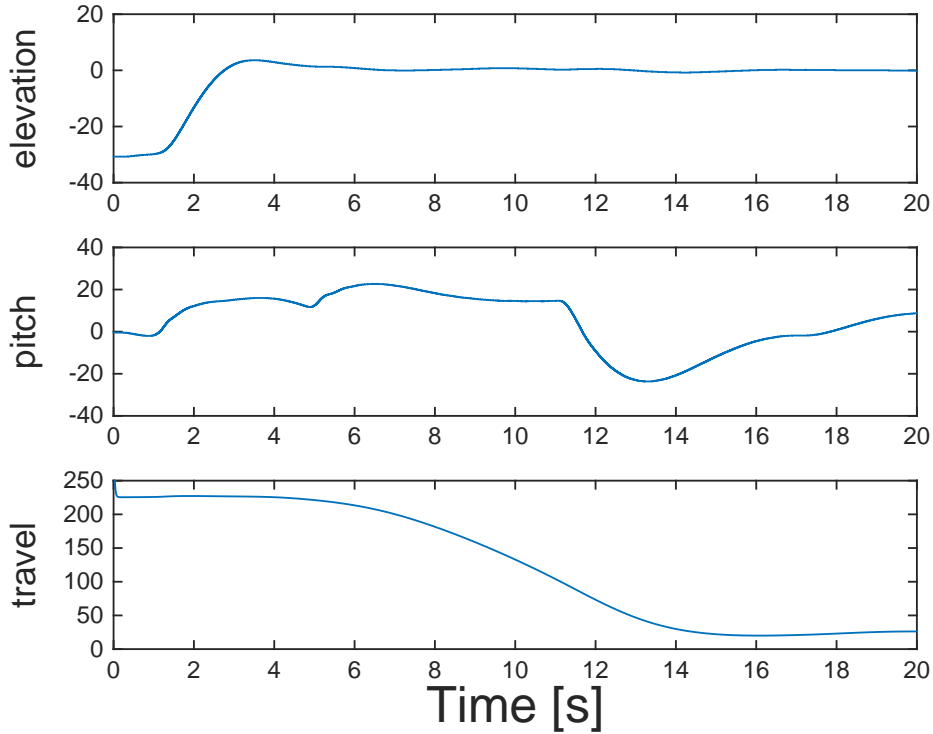


Figure 5: Running the helicopter with feedback

As shown the shape of both the travel and pitch match the calculated

values found in 1b well. The feedback distorts the pitch from the calculated optimal as it works in parallel with the optimal input for the helicopter.

### **4.3 MPC discussion (10.3.3)**

#### **4.3.1 Realization of MPC**

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

```
for  $t = 0, 1, 2, \dots$  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 of using MPC)**

There are both advantages and disadvantages with using an MPC controller compared to LQR. MPC requires a lot of computation, and may therefore require more expensive hardware or longer computation time. However, this can yield a very accurate controller. LQR is less computationally intensive, but also less accurate.

### 4.3.3 Figure 8 with MPC

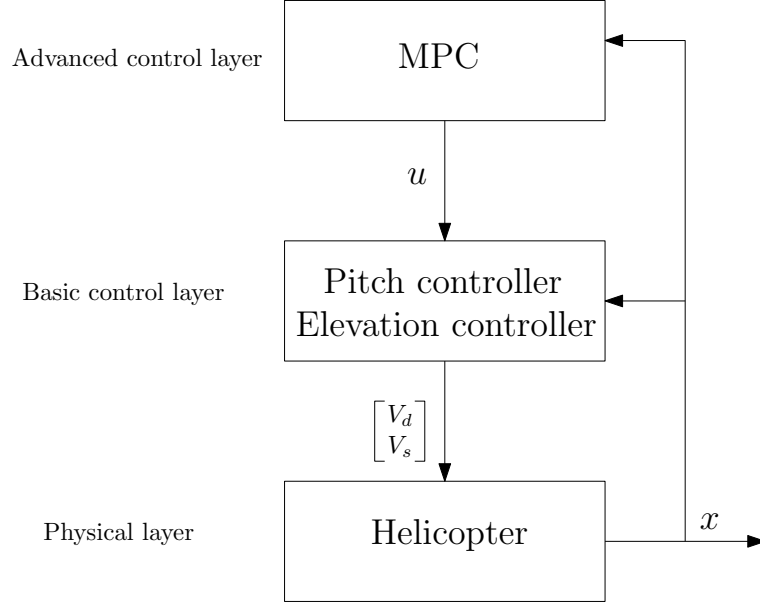


Figure 6: Figure 8 with MPC

The MPC, the pitch controller, and the elevation controller receives the measured 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) is expressed by the matrices shown in (21).

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \quad (18)$$

$$\mathbf{x} = [\lambda \quad r \quad p \quad \dot{p} \quad e \quad \dot{e}]^T, \quad \mathbf{u} = [p_c \quad e_c]^T \quad (19)$$

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

$$\dot{\mathbf{x}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3 K_{ep} & -K_3 K_{ed} \end{bmatrix}}_{\mathbf{A}_c} \mathbf{x} + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1 K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3 K_{ep} \end{bmatrix}}_{\mathbf{B}_c} \mathbf{u} \quad (21)$$

## 5.2 Discretisation again (10.4.2)

As in Section 3.3, we again need to discretize the system. The procedure is the same; refer to the earlier Section for clarification. The result can be seen in (23) and (24).

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

$$\mathbf{A} = \begin{bmatrix} 1 & h & 0 & 0 & 0 & 0 \\ 0 & 1 & -hK_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & h & 0 & 0 \\ 0 & 0 & -hK_1 K_{pp} & 1 - hK_1 K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & h \\ 0 & 0 & 0 & 0 & -hK_3 K_{ep} & 1 - hK_3 K_{ed} \end{bmatrix} \quad (23)$$

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ hK_1K_{pp} & 0 \\ 0 & 0 \\ 0 & hK_3K_{ep} \end{bmatrix} \quad (24)$$

### 5.3 Solution of the optimization problem (10.4.3)

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

$$\phi = \sum_{i=1}^N (\lambda_i - \lambda_f)^2 + q_1 p_{ci}^2 + q_2 e_{ci}^2 \quad (25)$$

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

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 used 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 initial values `z0`.

```
1 costf = @(z) 0.5*z'*Q*z;
2 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 avoid overshoot. With feedback, the helicopter will change its pitch to the opposite direction, making the helicopter stop and stay still at  $\lambda = \lambda_f$ .



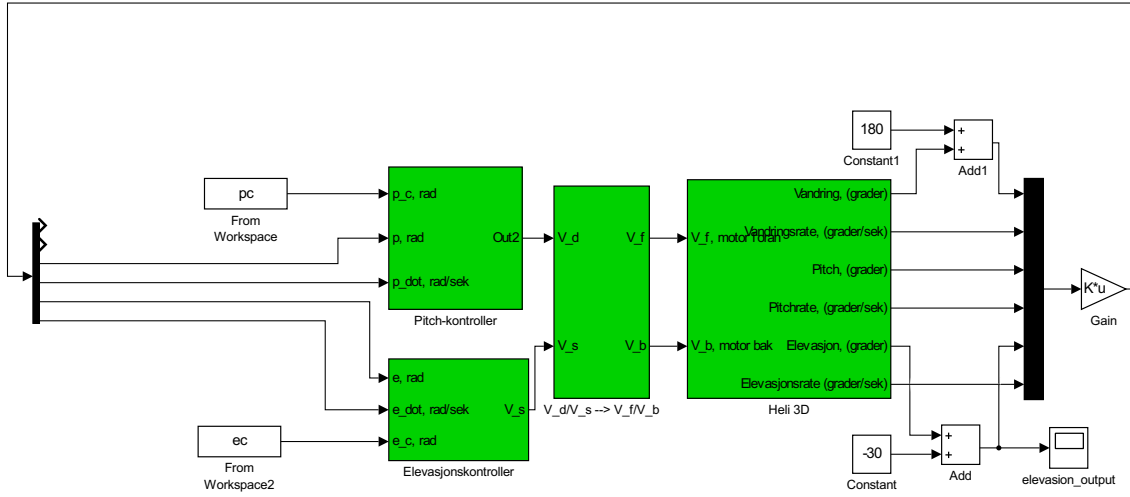


Figure 7: Simulink model of the system without feedback

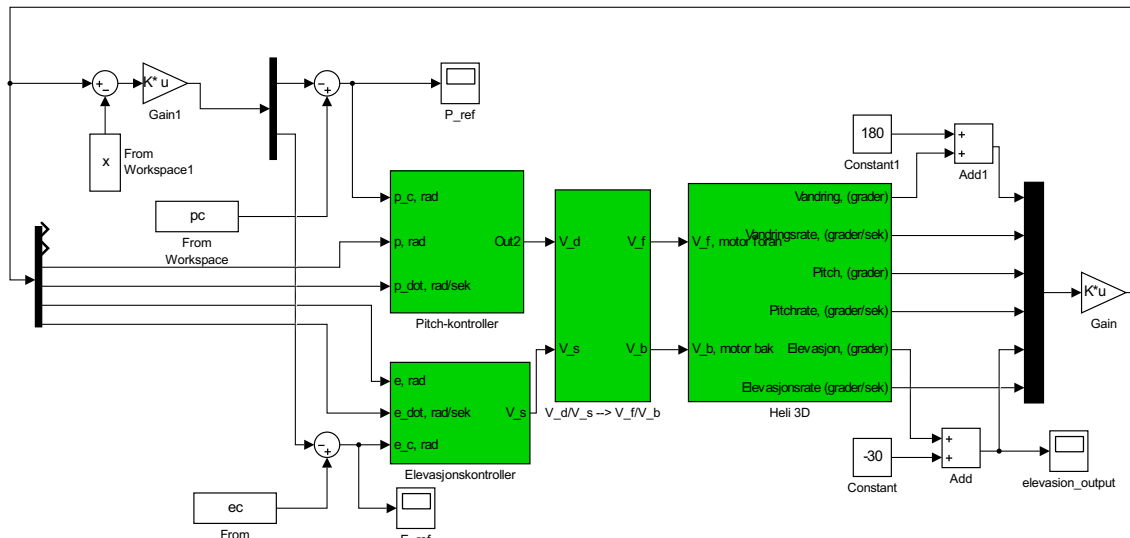


Figure 8: Simulink model of the system with feedback

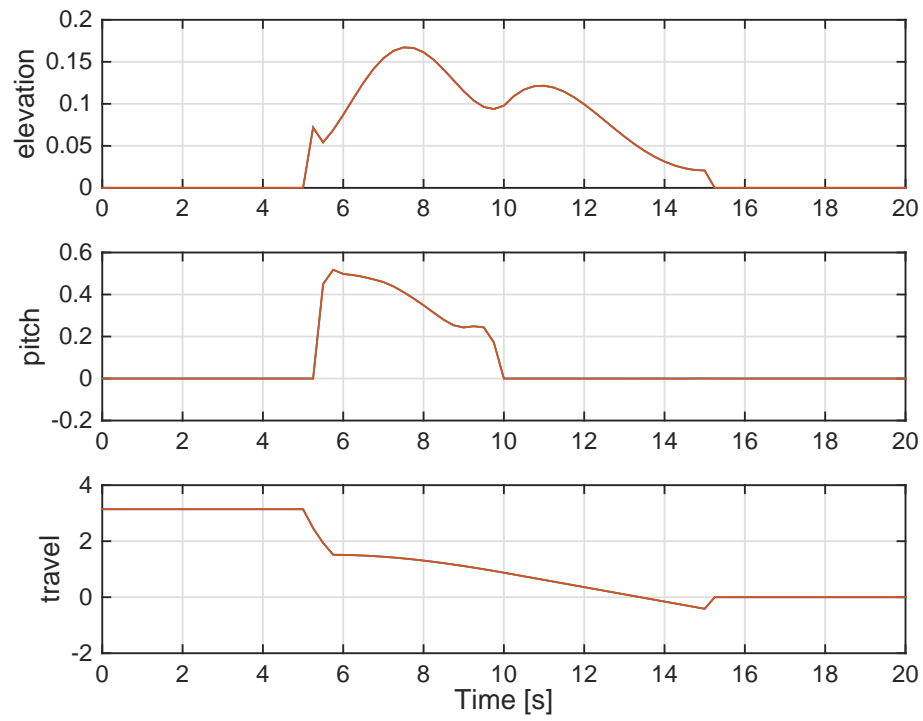


Figure 9: Optimal path

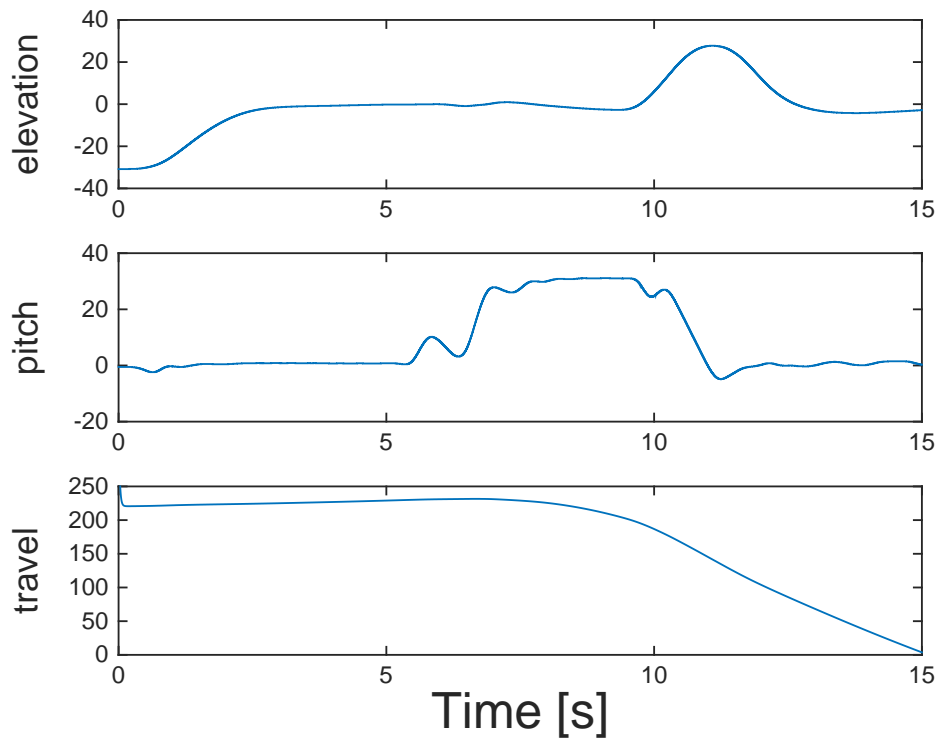


Figure 10: Running the helicopter without feedback

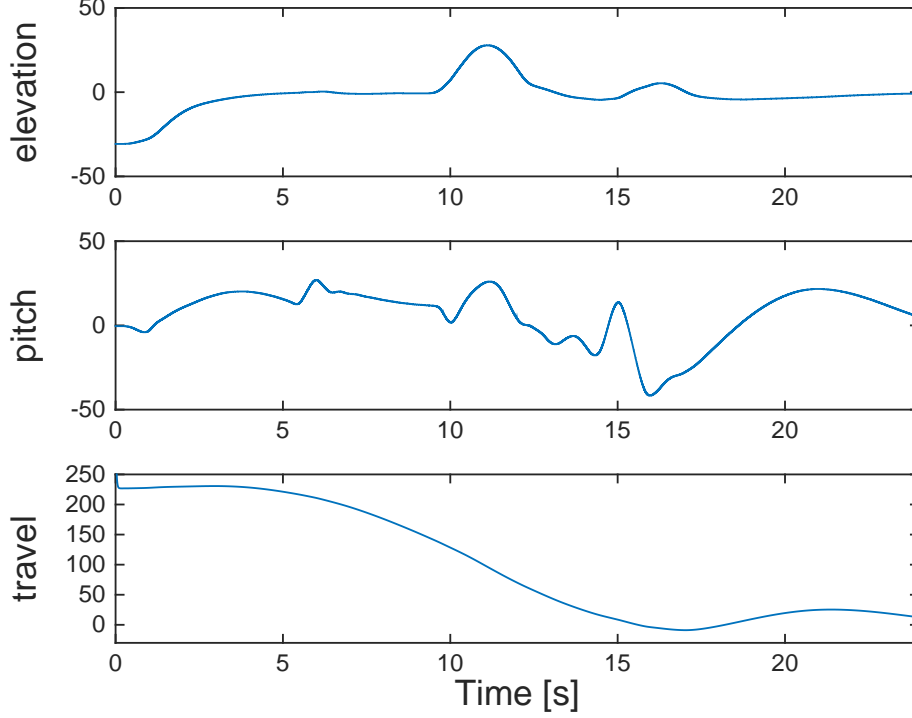


Figure 11: Running the helicopter with feedback

As we can see in the sensor data from the model without feedback, the helicopter does not stop at  $\lambda = \lambda_f$ , but keeps moving. 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 roughly 16 seconds. It also overshoots slightly, but converges at the setpoint.

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 the real behaviour, especially with feedback. This would correspond better if the LQR was tuned better (how the LQR was tuned can be seen in the code in the appendix). In the measured 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 improved in the model with feedback, although it does not stop as quickly and smoothly as the calculated output would suggest.

## 5.5 Discussion of decoupled states (10.4.5)

In our model the elevation and the elevation rate are completely decoupled from the other states, which isn't really realistic. 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 result in an offset from 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

We feel that it has been interesting to work on this lab, and that we have learnt a lot by doing it. Through a practical approach to implementing optimization algorithms, we have gained insight into how it might 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, and it has been both rewarding and educational.

## 8 Appendix

### 8.1 Variables

Symbol	Variable
$p$	Pitch
$p_c$	Pitch setpoint
$\lambda$	Travel
$r$	Travel rate
$r_c$	Travel rate setpoint
$e$	Elevation
$e_c$	Elevation setpoint
$V_f$	Voltage, front motor
$V_b$	Voltage, rear motor
$V_d$	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 required 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	N m <sup>-1</sup>
$J_e$	Moment of inertia (elevation axis)	1.0625	kg m <sup>2</sup>
$J_t$	Moment of inertia (travel axis)	1.0625	kg m <sup>2</sup>
$J_p$	Moment of inertia (pitch axis)	0.0406	kg m <sup>2</sup>
$m_h$	Mass of helicopter	1.297	kg
$m_w$	Counterweight	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 modified from the code handed out on itslearning.

```

1 init04;
2 delta_t = 0.25; % sampling time
3 sek_forst = 5;
4
5 q = 0.1;
6 % System model. x=[lambda r p p_dot]'
7
8 A1 = [1 delta_t 0 0;
9       0 1 -K_2*delta_t 0;
10      0 0 1 delta_t;
11      0 0 -K_1*K_pp*delta_t 1-K_1*K_pd*delta_t];
12
13 B1 = [0; 0; 0; K_1*K_pp*delta_t];
14
15 % Number of states and inputs
16
17 mx = size(A1,2); % Number of states
18 mu = size(B1,2); % Number of inputs
19
20 % Initial values
21
22 x1_0 = pi; % Lambda
23 x2_0 = 0; % r
24 x3_0 = 0; % p
25 x4_0 = 0; % p_dot
26 x0 = [x1_0 x2_0 x3_0 x4_0]'; % Initial values
27
28 % Time horizon and initialization
29
30 N = 100; % Time horizon for states
31 M = N; % Time horizon for inputs
32 z = zeros(N*mx+M*mu,1); % Initialize z for whole horizon
33 z0 = z; % Initial value for optimization
34
35 % Bounds
36
37 ul = -30*pi/180; % Lower bound on control -- ul
38 uu = 30*pi/180; % Upper bound on control -- ul
39
40 x1 = -Inf*ones(mx,1); % Lower bound on states (no bound)
41 xu = Inf*ones(mx,1); % Upper bound on states (no bound)
42 x1(3) = ul; % Lower bound on state x3
43 xu(3) = uu; % Upper bound on state x3
44
45 % Generate constraints on measurements and inputs
46
47 [v1b,vub] = genBegr2(N,M,x1,xu,ul,uu);
48 v1b(N*mx+M*mu) = 0; % Last input is zero

```

```

49 vub(N*mx+M*mu) = 0; % Last input is zero
50
51 % Generate the matrix Q and the vector c (objective function
    ↳ weights in the QP problem)
52
53 Q1 = zeros(mx,mx);
54 Q1(1,1) = 1; % Weight on state x1
55 Q1(2,2) = 0; % Weight on state x2
56 Q1(3,3) = q; % Weight on state x3
57 Q1(4,4) = 0; % Weight on state x4
58 P1 = q; % Weight on input
59 Q = 2*genq2(Q1,P1,N,M,mu); % Generate Q
60 c = zeros(N*mx+M*mu,1); % Generate c
61
62 % Generate system matrixes for linear model
63
64 Aeq = gena2(A1,B1,N,mx,mu); % Generate A
65 beq = zeros(1, size(Aeq,1)); % Generate b
66 beq(1:mx) = A1*x0; % Initial value
67
68 % Solve Qp problem with linear model
69 tic
70 [z,lambda] = quadprog(Q, c, [], [], Aeq, beq, vlb, vub, z0);
71 t1=toc;
72
73
74 % Extract control inputs and states
75
76 u = [z(N*mx+1:N*mx+M*mu); z(N*mx+M*mu)]; % Control input from
    ↳ solution
77
78 x1 = [x0(1); z(1:mx:N*mx)]; % State x1 from solution
79 x2 = [x0(2); z(2:mx:N*mx)]; % State x2 from solution
80 x3 = [x0(3); z(3:mx:N*mx)]; % State x3 from solution
81 x4 = [x0(4); z(4:mx:N*mx)]; % State x4 from solution
82
83 Antall = 5/delta_t;
84 Nuller = zeros(Antall,1);
85 Enere = ones(Antall,1);
86
87 u = [Nuller; u; Nuller];
88 x1 = [pi*Enere; x1; Nuller];
89 x2 = [Nuller; x2; Nuller];
90 x3 = [Nuller; x3; Nuller];
91 x4 = [Nuller; x4; Nuller];
92
93 %save trajektorlmy
94 t = 0:delta_t:delta_t*(length(u)-1); % real time
95

```



```

96 simin = [t' u];
97
98 % figure
99
100
101 figure(2)
102 subplot(511)
103 stairs(t,u),grid
104 ylabel('u')
105 subplot(512)
106 plot(t,x1,'m',t,x1,'mo'),grid
107 ylabel('lambda')
108 subplot(513)
109 plot(t,x2,'m',t,x2,'mo'),grid
110 ylabel('r')
111 subplot(514)
112 plot(t,x3,'m',t,x3,'mo'),grid
113 ylabel('p')
114 subplot(515)
115 plot(t,x4,'m',t,x4,'mo'),grid
116 xlabel('tid (s)'),ylabel('p_dot')
117
118 x = [t' x1 x2 x3 x4];
119 %%
120
121 Q_k = [25 0 0 0;
122        0 0.5 0 0;
123        0 0 100 0;
124        0 0 0 0.5];
125 R = 1;
126
127 [K S E] = dlqr(A1, B1, Q_k, R, 0);

```

## 8.4 Code for (10.4)

This code is modified from the code handed out on itslearning.

```

1 init04;
2 delta_t = 0.25; % sampling time
3 sek_forst = 5;
4
5 q = 0.1;
6 % System model. x=[lambda r p p_dot]'
7
8 A = [1 delta_t 0 0 0 0;
9      0 1 -K_2*delta_t 0 0 0;
10     0 0 1 delta_t 0 0;

```

```

11         0 0 -K_1*K_pp*delta_t 1-K_1*K_pd*delta_t 0 0;
12         0 0 0 0 1 delta_t;
13         0 0 0 0 -delta_t*K_3*K_ep 1-delta_t*K_3*K_ed];
14
15 B = [0 0; 0 0; 0 0; K_1*K_pp*delta_t 0; 0 0; 0 delta_t*K_3*K_ep
    ↪ ];
16
17 % Number of states and inputs
18
19 mx = size(A,2); % Number of states
20 mu = size(B,2); % Number of inputs
21
22 % Initial values
23
24 x1_0 = pi; % Lambda
25 x2_0 = 0; % r
26 x3_0 = 0; % p
27 x4_0 = 0; % p_dot
28 x5_0 = 0; % e
29 x6_0 = 0; % e_dot
30 x0 = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0]'; % Initial
31
32 u1_0 = 0;
33 u2_0 = 0;
34
35 q_1 = 1;
36 q_2 = 1;
37 alfa = 0.2;
38 beta = 20;
39 lambda_t = 2*pi/3;
40
41 % Time horizon and initialization
42
43 N = 40; % Time horizon for states
44 M = N; % Time horizon for inputs
45 z = zeros(N*mx+M*mu,1); % Initialize z for whole horizon
46 z0 = z; % Initial value for optimization
47
48
49 % Bounds
50
51 ul = -30*pi/180; % Lower bound on control -- u1
52 uu = 30*pi/180; % Upper bound on control -- u1
53
54 xl = -Inf*ones(mx,1); % Lower bound on states (no bound)
55 xu = Inf*ones(mx,1); % Upper bound on states (no bound)
56 xl(3) = ul; % Lower bound on state x3
57 xu(3) = uu; % Upper bound on state x3
58

```

```

59 % Generate constraints on measurements and inputs
60
61 [v1b,vub] = genBegr2(N,M,x1,xu,u1,uu);
62 v1b(N*mx+M*mu) = 0; % Last input is zero
63 vub(N*mx+M*mu) = 0; % Last input is zero
64
65 % Generate system matrixes for linear model
66 Aeq = gena2(A,B,N,mx,mu); % Generate A
67 beq = zeros(1, size(Aeq,1)); % Generate b
68 beq(1:mx) = A*x0; % Initial value
69
70 % Generate the matrix Q and the vector c (objecitve function
    ↪ weights in the QP problem)
71
72 Q1 = zeros(mx,mx);
73 Q1(1,1) = 1; % Weight on state x1
74 Q1(2,2) = 0; % Weight on state x2
75 Q1(3,3) = q; % Weight on state x3
76 Q1(4,4) = 0; % Weight on state x4
77 Q1(5,5) = 0;
78 Q1(6,6) = 0;
79 P1 = zeros(mu, mu);
80 P1(1,1) = q_1; % Weight on input
81 P1(2,2) = q_2;
82 Q = 2*genq2(Q1,P1,N,M,mu); % Generate Q
83 c = zeros(N*mx+M*mu,1); % Generate c
84
85 % Solve Qp problem with linear model
86
87 costf = @(z) 0.5*z'*Q*z;
88 %tic
89 %[z,lambda] = quadprog(Q, c, [], [], Aeq, beq, v1b, vub, z0);
90 %t1=toc;
91 nonlcon = @(z) alfa*exp(-beta*(z(1+mx)-lambda_t)^2)-z(5+mx);
92 z = fmincon(costf, z0, [], [], Aeq, beq, v1b, vub, @mycon);
93 % Calculate objective value
94
95 phil = 0.0;
96 PhiOut = zeros(N*mx+M*mu,1);
97 for i=1:N*mx+M*mu
98     phil=phil+Q(i,i)*z(i)*z(i);
99     PhiOut(i) = phil;
100 end
101
102 % Extract control inputs and states
103
104 u1 = [u1_0; z(N*mx+1:mu:N*mx+M*mu)]; % Control input from
    ↪ solution
105 u2 = [u2_0; z(N*mx+2:mu:N*mx+M*mu)];

```

```

106
107 x1 = [x0(1); z(1:mx:N*mx)]; % State x1 from solution
108 x2 = [x0(2); z(2:mx:N*mx)]; % State x2 from solution
109 x3 = [x0(3); z(3:mx:N*mx)]; % State x3 from solution
110 x4 = [x0(4); z(4:mx:N*mx)]; % State x4 from solution
111 x5 = [x0(5); z(5:mx:N*mx)];
112 x6 = [x0(6); z(6:mx:N*mx)];
113
114 Antall = 5/delta_t;
115 Nuller = zeros(Antall,1);
116 Enere = ones(Antall,1);
117
118 u1 = [Nuller; u1; Nuller];
119 u2 = [Nuller; u2; Nuller];
120
121 x1 = [x1_0*Enere; x1; Nuller];
122 x2 = [Nuller; x2; Nuller];
123 x3 = [Nuller; x3; Nuller];
124 x4 = [Nuller; x4; Nuller];
125 x5 = [Nuller; x5; Nuller];
126 x6 = [Nuller; x6; Nuller];
127
128 %save trajektorlmy
129 t = 0:delta_t:delta_t*(length(u1)-1); % real time
130
131 pc = [t' u1];
132 ec = [t' u2];
133 % figure
134
135
136 figure(2)
137 subplot(8,1,1)
138 stairs(t,u1),grid
139 ylabel('u1')
140 subplot(8,1,2)
141 stairs(t, u2),grid
142 ylabel('u2')
143 subplot(8,1,3)
144 plot(t,x1,'m',t,x1,'mo'),grid
145 ylabel('lambda')
146 subplot(8,1,4)
147 plot(t,x2,'m',t,x2,'mo'),grid
148 ylabel('r')
149 subplot(8,1,5)
150 plot(t,x3,'m',t,x3,'mo'),grid
151 ylabel('p')
152 subplot(8,1,6)
153 plot(t,x4,'m',t,x4,'mo'),grid
154 ylabel('pdot')

```

```

155 subplot(8,1,7)
156 plot(t,x5,'m',t,x5','mo'),grid
157 ylabel('e')
158 subplot(8,1,8)
159 plot(t,x6,'m',t,x6','mo'),grid
160 ylabel('edot')
161 xlabel('tid (s)')
162
163 x = [t' x1 x2 x3 x4 x5 x6];
164
165 %%
166 Q_k = [2 0 0 0 0 0;
167        0 1 0 0 0 0;
168        0 0 1 0 0 0;
169        0 0 0 1.2 0 0;
170        0 0 0 0 1 0;
171        0 0 0 0 0 1.5];
172
173 R = [1 0;
174      0 1];
175
176 [K S E] = dlqr(A, B, Q_k, R, 0);

```

Code for the non-linear constrains

```

1 function [a, b] = mycon(x)
2
3 % system parameters
4 alpha = 0.2;
5 beta = 20;
6 lambda_t = 2*pi/3;
7 N = 40;
8
9 myc = @(lambda, e) alpha * exp(-beta .* (lambda - lambda_t).^2)
10    ↪ - e;
11
12 c = @(x) myc(x(1 : 6 : 6*N), x(5 : 6 : 6*N));
13
14 a = c(x);
15 b = [];
16
17 end

```