Helicopter Lab report

TTK4135 - Optimization and Control

Group 39:
742003
748667

NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Engineering Cybernetics
NTNU, Norway
March 2017

Abstract

Optimal trajectories between two points will be calculated based on a linearized model for a mounted helicopter with two actuators. An LQR is implemented in the advanced control layer to minimize undesired deviations. The possible use of an MPC is discussed.

Contents

Introduction

The goal of the project is to calculate an optimal route for a mounted helicopter with and without feedback using an linear quadratic controller. A non-linear model will be linearized around equilibrium and trajectories will be created by minimizing the cost function using built in Matlab functions for simplicity. Optimal trajectories will be tuned and tested based on different state weights, and mathematical models and choices for regulator optimizations and constraints will be thoroughly derived.

The report is organized as follows: Section 1 contains a description of the lab setup including mathematical modeling describing the system dynamics. In section 2 optimal control of the travel trajectory without feedback is introduced. Section 3 introduces an advance control layer in the form of a LQR, regulating the optimal input and trajectories calculated in section 2. In section 4 an optimal elevation trajectory is introduced, including a non-linear inequality constraint yielding a non-linear optimization problem in two dimensions. Appendix A contains diagrams of the relevant Simulink models and appendix B contains MATLAB scripts.

1 Problem Description

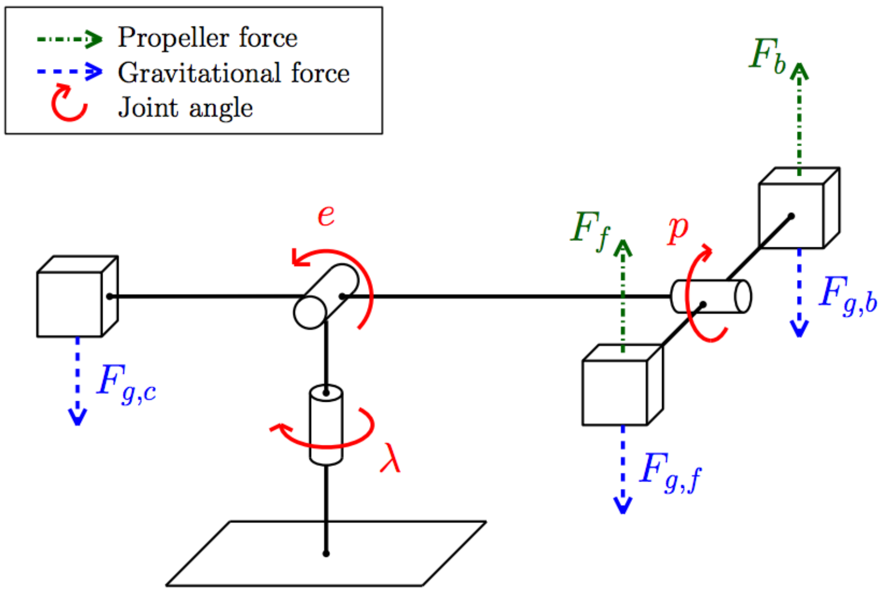

Figure 1: Helicopter model

The hardware consists of a mounted helicopter with two actuators and a counterweight. The lift generated by the actuators depends on the power delivered to each actuator. If the corresponding power between the actuators is not equal, a spin around joint p will be generated, changing the angle of the force provided by the actuators. A change in force angle will create a spin around joint λ, which provides a gain in both travel and travel-rate.

Simple differential equations describing the systems dynamics around the equilibrium can be found by formulating the moment balances. [1].

$$\ddot{e} = K_3 V_s - \frac{T_g}{J_e}, \quad K_3 = \frac{l_a K_f}{J_e} \tag{1a}$$

$$\ddot{p} = K_1 V_d, \quad K_1 = \frac{K_f l_h}{J_p} \tag{1b}$$

$$\dot{r} = -K_2 p, \quad K_2 = \frac{K_p l_a}{J_t} \tag{1c}$$

Adding a PD-controller to (1a)

$$V_s = K_{ep}(e_c - e) - K_{ed}\dot{e}, \quad K_{ep}, K_{ed} > 0 \tag{2a}$$

and including an integral term, effectively removing $-\frac{T_g}{J_e}$, yields

$$\ddot{e} + K_3 K_{ed}\dot{e} + K3Kepe = K_3 K_{ep} e_c \tag{3}$$

A PD controller

$$V_d = K_{pp}(p_c - p) - K_{pd}\dot{p}, \quad K_{pp}, K_{pd} > 0 \tag{4}$$

is implemented to control the pitch angle. These yields the model equations

$$\ddot{e} + K_3 K_{ed}\dot{e} + K_3 K_{ep} e = K_3 K_{ep} e_c \tag{5a}$$
$$\ddot{p} + K_1 K_{pd}\dot{p} + K_1 K_{pp} p = K_1 K_{pp} p_c \tag{5b}$$
$$\dot{\lambda} = r \tag{5c}$$
$$\dot{r} = -K_2 p \tag{5d}$$

With parameters and values as defined in *Table 1* and *Table 2* [1]. All angles in the equations above are given in radians.

Table 1: Parameters and values.

Symbol	Parameter	Value	Unit
l_a	Distance from elevation axis to helicopter body	0.63	m
l_h	Distance from pitch axis to motor	0.18	m
K_f	Force constant motor	0.25	N/V
J_e	Moment of inertia for elevation	0.83	$\mathrm{kg\,m^2}$
J_t	Moment of inertia for travel	0.83	$\mathrm{kg\,m^2}$
J_p	Moment of inertia for pitch	0.034	$\mathrm{kg\,m^2}$
m_h	Mass of helicopter	1.05	kg
m_w	Balance weight	1.87	kg
m_g	Effective mass of the helicopter	0.05	kg
K_p	Force to lift the helicopter from the ground	0.49	N

Table 2: 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

2 Optimal Control of Pitch/Travel without Feedback

2.1 Continous state space form

Given the simplified model without including elevation, $x = \begin{bmatrix}\lambda & r & p & \dot{p}\end{bmatrix}^T, u = p_c$, the continuous state space form is derived using equation 5b, 5c and 5d

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} \end{bmatrix} \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1K_{pp} \end{bmatrix} \begin{bmatrix} p_c \end{bmatrix} \tag{6}$$

The model includes travel, travel rate, pitch and pitch rate. Input u^* is the calculated optimal input sequence. A simple overview can be seen in figure 2 [1].

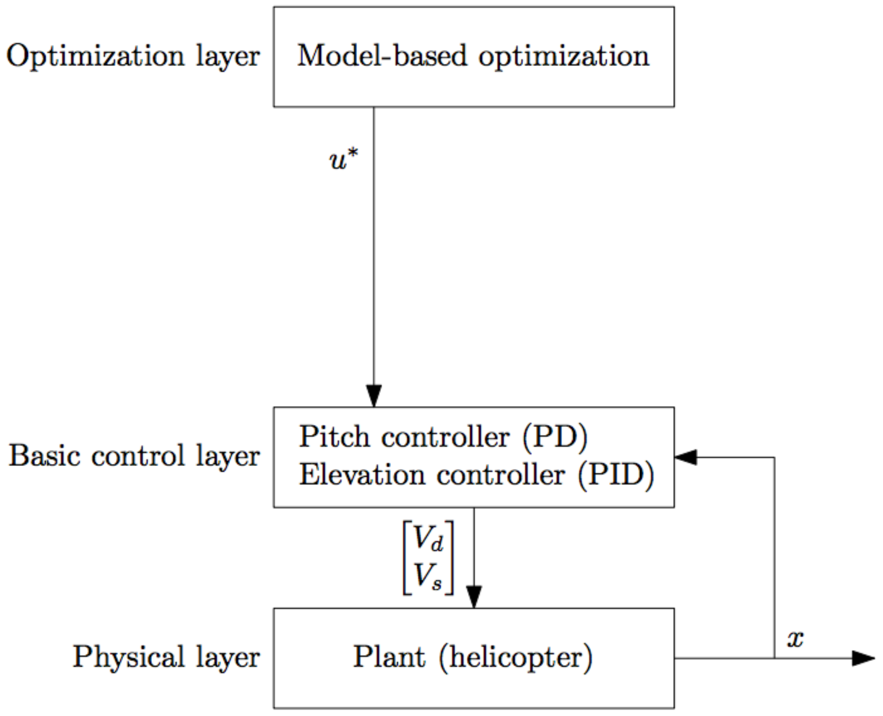

Figure 2: Illustration of the layers in the control hierarchy used in Section 2

2.2 Discretization

Discretization is the first step towards making the continuous process suitable for numerical evaluation and implementation on digital computers [2]. A discrete model was calculated using Eulers method with a time step of $\triangle t = 0.25s$

$$\begin{aligned} x_{k+1} &= x_k + \triangle t \dot{x}_k \\ &= x_k + \triangle t(A_c x_k + B_c u_k) \\ &= (I + \triangle t A_c)x_k + (\triangle t B_c)u_k \\ &= Ax_k + Bu_k \end{aligned} \tag{7}$$

Which results in

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 1 & 0.2500 & 0 & 0 \\ 0 & 0 & -0.1415 & 0 \\ 0 & 0 & 1 & 0.25 \\ 0 & 0 & -0.8100 & 1.000 \end{bmatrix} \begin{bmatrix} \lambda \\ 0 \\ 0 \\ \dot{p} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.8100 \end{bmatrix} \begin{bmatrix} p_c \end{bmatrix} \tag{8}$$

Euler method is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value [2]. The method has become quite attractive because of its simplicity of implementation.

2.3 Optimal trajectory

An optimal trajectory for moving the helicopter from $x_0 = \begin{bmatrix} \pi & 0 & 0 & 0 \end{bmatrix}^T$ to $x_f = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$ can be found by minimizing the cost function

$$\phi = \sum_{i=1}^{N} (\lambda_i - \lambda_f)^2 + r p_{c_i - 1}^2, r > 0 \tag{9}$$

The solution for the optimization problem was found using the MATLAB function *quadprog*, as shown in appendix B, section B.2.

$$x = quadprog(H, f, A, b, Aeq, beq) \tag{10}$$

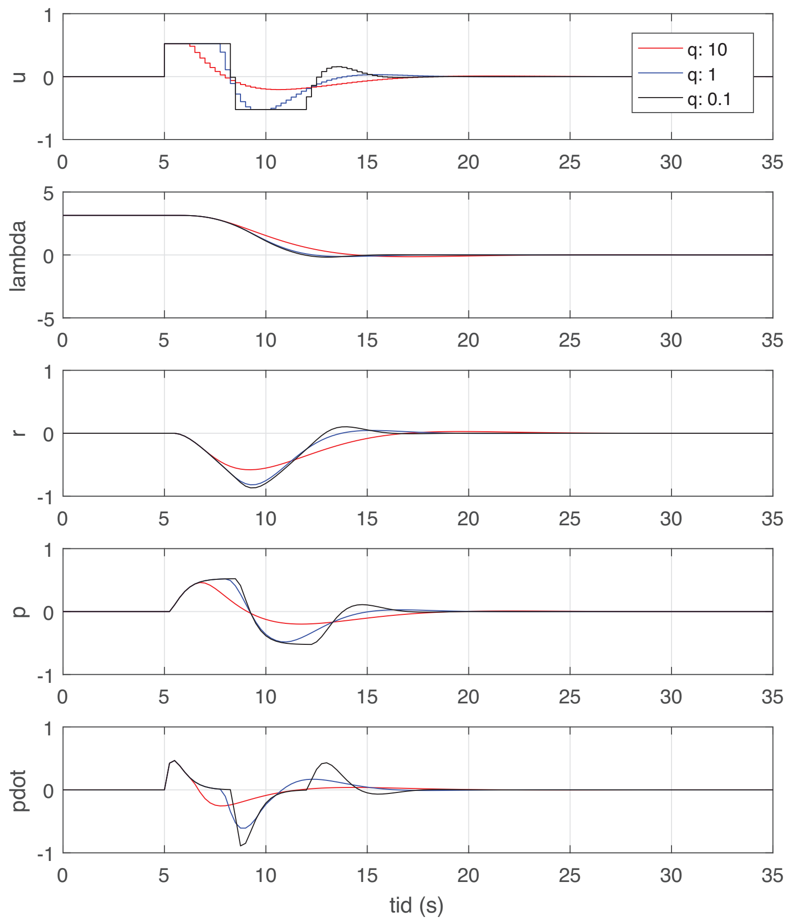

Figure 3: Calculated optimal trajectories with different weights of x_1

As you can see in figure 3, the optimal trajectory is affected on how you weight the different states. Based on changing the weight on the input signal u, $q = 10$ provides a slow convergence towards the reference. In the other end, $q = 0.1$ creates fluctuations on the optimal pitch trajectory, which may cause damage to the hardware after implementation. The more fluctuations on the optimal trajectory, the more difficult it will be for the regulator to follow the calculated route, to a point where the system may fail. $q = 1$ is an optimal value for our system, the configuration makes the helicopter converge rapidly towards the reference without causing too much fluctuations and we maximize the pitch between the upper and lower bounds. The following Q and R matrices are used throughout the exercise, except section 4

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} R = \begin{bmatrix} 1 \end{bmatrix} \tag{11}$$

2.4 Implementation of optimal trajectory

Under the testing of the optimal trajectory from $x_0 = \begin{bmatrix} \pi & 0 & 0 & 0 \end{bmatrix}^T$ to $x_f = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$ the operations went smooth with some deviations. The helicopter followed the calculated route, but went past the destination. The system is not implemented with feedback on travel λ and travel rate r, so this is to be expected. Deviations on these variables will not be corrected and may increase if you keep the system online. This does not apply to the pitch which is implemented with an inner controller.

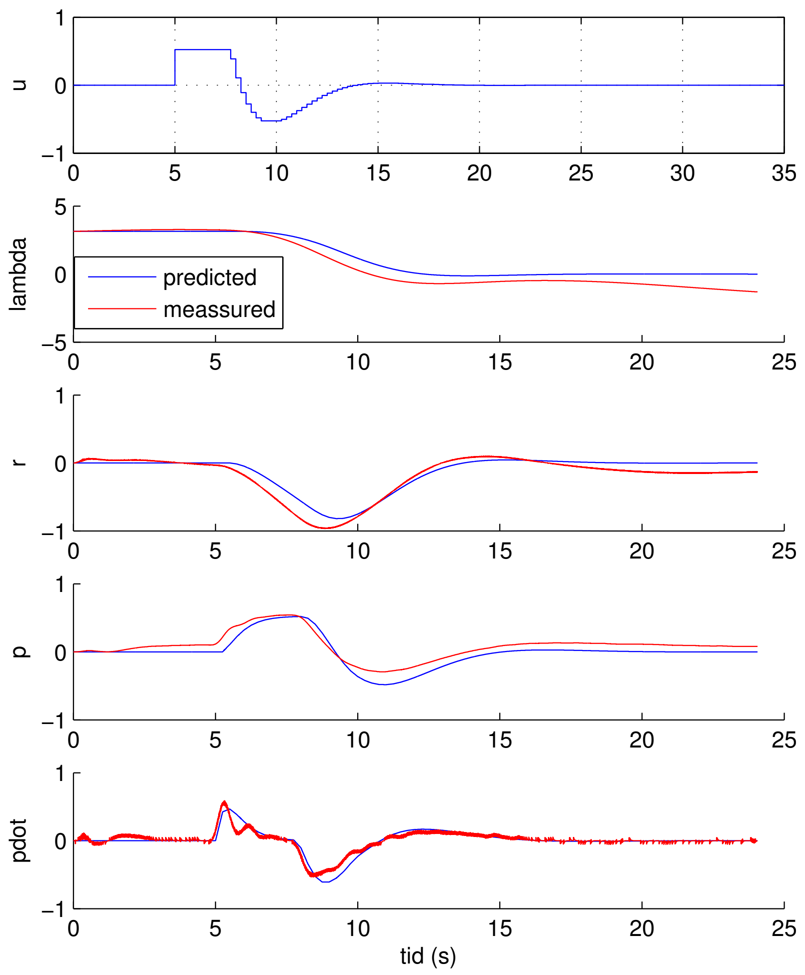

Figure 4: Actual measurements compared to optimal trajectories without feedback

3 Optimal Control of Pitch/Travel with Feedback

3.1 Introduction of feedback in the optimal controller

Feedback is introduced to the optimal controller through the use of a linear quadratic controller. An LQ-controller (in discrete time) minimizes the quadratic objective function

$$J = \sum_{i=0}^{\infty} \Delta x_{i+1}^{\top} Q \Delta x_{i+1} + \Delta u_i^{\top} R \Delta u_i \quad Q \geq 0, R > 0 \tag{12}$$

for a linear model

$$\Delta x_{i+1} = A \Delta x_i + B \Delta u_i \tag{13}$$

without including inequality constraints.

The following manipulated variable is an result of the optimal input sequence u^* combined with feedback

$$u_k = u_k^* - K^{\top}(x_k - x_k^*) \tag{14}$$

As long as the system is following the calculated optimal trajectory x^*, the optimal input sequence u^* will be used without manipulation. If a deviation from the optimal trajectory is observed, the feedback

$$K^{\top}(x_k - x_k^*) \tag{15}$$

will correct the system from the deviated course.

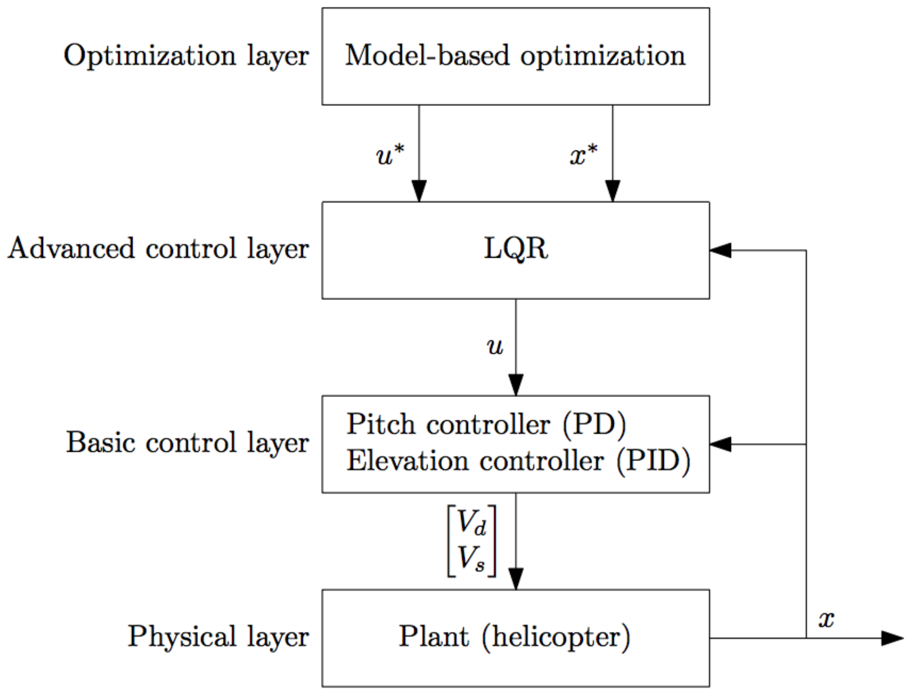

Figure 5: Illustration of layers in the control hierarchy used in Section 3

The gain matrix K can be calculated using the Matlab-function *dlqr*[6].

$$[K, S, e] = dlqr(A, B, Q, R, N) \tag{16}$$

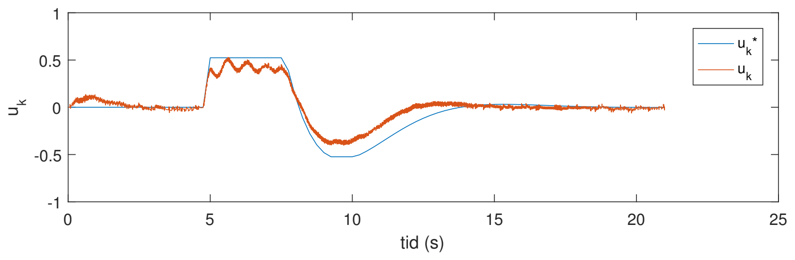

Figure 6: Optimal input sequence compared with and without feedback

As observed in figure 6, the optimal input sequence changes based on state feedback. Equation 14 ensures that a new optimal pitch reference p is calculated at each time step. The result is discussed in section 3.2

3.2 Implementation of the feedback on the helicopter

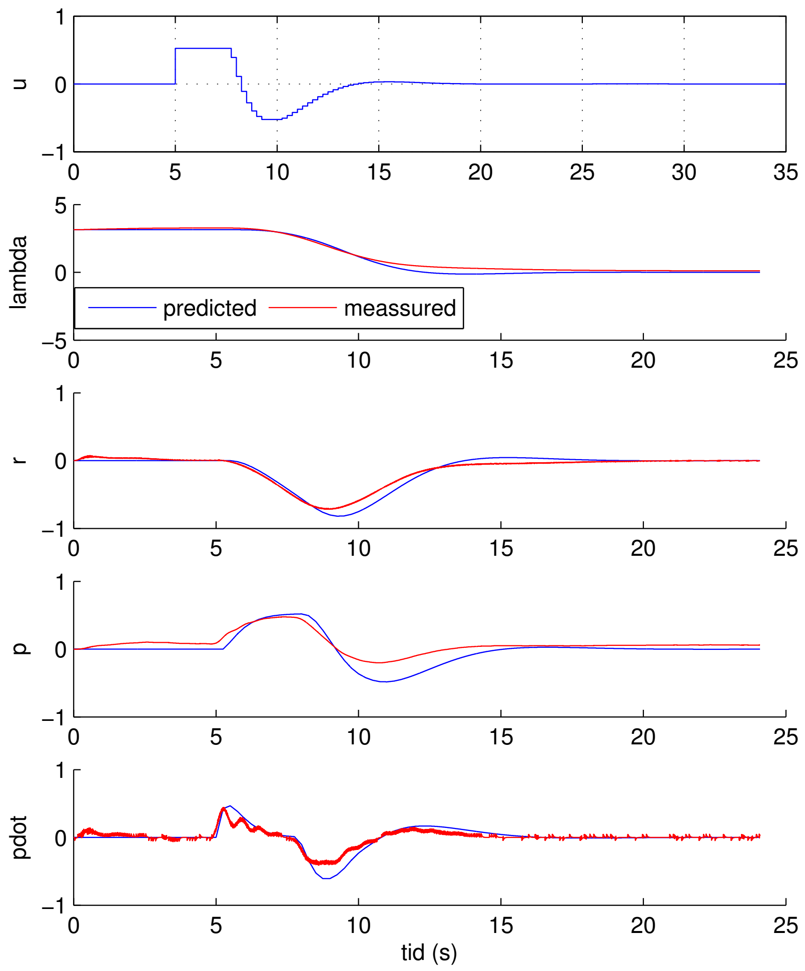

Figure 7: Actual measurements compared to optimal trajectories with feedback

As seen in figure 7, the system is way more accurate with feedback implemented. The helicopter follows the optimal trajectory and corrects deviations from the calculated route. Note that system almost precisely follows the travel, λ, which is the most important variable. The main purpose of the assignment is to guide the helicopter from $\lambda = pi$ to $\lambda = 0$, and the LQR Q and R matrixes has to be defined based on that fact, punishing deviations in travel. See appendix, plot 12 for the Simulink representation.

$$Q_{LQR} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_{LQR} = \begin{bmatrix} 0.1 \end{bmatrix} \tag{17}$$

3.3 Alternative strategy with an MPC controller

Model predictive control solves a finite horizon optimal control problem at each time step. An MPC can be implemented using two methods, one with actual feedback using the current state, and one calculating the current state using all previous measurements [4]. One problem with first method is inaccurate system states at any time step, the disturbances will affect the quality of the route calculated with the model predictive controller. Another problem is choosing the initial value, which may not be correct because it does not account for errors in the discrete time model and disturbances that occur between t and t+1. In general, without state disturbances, you will achieve a more optimal control as you recalculate at every step.

MPC has the ability to anticipate future events and can take control actions accordingly. This features can be a huge advantage in control systems, as long as you have a simple system and a short time horizon.

Implementation may not be a good solution, as in our case. A new optimal input sequence would be received too late. A MPC demands powerful hardware as each time-step includes calculations which comes at a high computational cost. However, an MPC should be implemented if possible.

4 Optimal Control of Pitch/Travel and Elevation with and without Feedback

The task is to calculate an optimal trajectory in two dimensions. The helicopter is moved from x_0 to x_f past a restriction causing the elevation angle to change during the flight. Elevation must therefore be added to the state vector. $\mathbf{x}$ is given by $\begin{bmatrix} \lambda & r & p & \dot{p} & e & \dot{e} \end{bmatrix}^T$, where e_c is the new manipulated variable in the system. This gives u = $\begin{bmatrix} p_c & e_c \end{bmatrix}^T$. [1]

4.1 Continuous state space form

The system can be written on continuous state space form, using equation 5. The resulting matrix is given below:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3K_{ep} & -K_3K_{ed} \end{bmatrix} \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \\ e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3K_{ep} \end{bmatrix} \begin{bmatrix} p_c \\ e_c \end{bmatrix} \tag{18}$$

4.2 Discrete state space form

Using Euler discretization described by equation 7, the resulting state space model is given as:

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 1 & 0.25 & 0 & 0 & 0 & 0 \\ 0 & 1 & -0.1415 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.2500 & 0 & 0 \\ 0 & 0 & -0.8100 & 0.100 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0.2500 \\ 0 & 0 & 0 & 0 & -0.0625 & 0.7500 \end{bmatrix} \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \\ e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0.8100 & 0 \\ 0 & 0 \\ 0 & 0.0625 \end{bmatrix} \begin{bmatrix} p_c \\ e_c \end{bmatrix} \tag{19}$$

4.3 Inequality constraints

The inequality constraints on elevation can be implemented

$$e_k \geq \alpha exp(-\beta(\lambda_k - \lambda_t)^2) \quad \forall k \ \epsilon \ \{1, ..., N\} \tag{20}$$

using equation 20 as shown in B.9. This is a nonlinear constraint, indicating that a standard QP-solver is not sufficient for solving this problem. Using the nonlinear programming solver **fmincon** [5] and the active-set method as shown in B.7, the cost function

$$\phi = \sum_{i=1}^{N} (\lambda_i - \lambda_f)^2 + q_1 p_{c_i - 1}^2 + q_2 e_{c_i}^2, \quad q_1, q_2 > 0 \tag{21}$$

can be minimized, and the resulting optimal trajectory can be found. For every point in time a constraint on the form

$$c(x_k) = \alpha exp(-\beta(\lambda_k - \lambda_t)^2) - e_k \tag{22}$$

must be implemented and passed fo fmincon.

A relative short time horizon ($N = 40$) is used because optimization over longer horizons correlates to a higher calculation time. α is set to 0.5, increasing the height of the constraint. $\beta = 20$ and $\lambda_t = \frac{2\pi}{3}$.

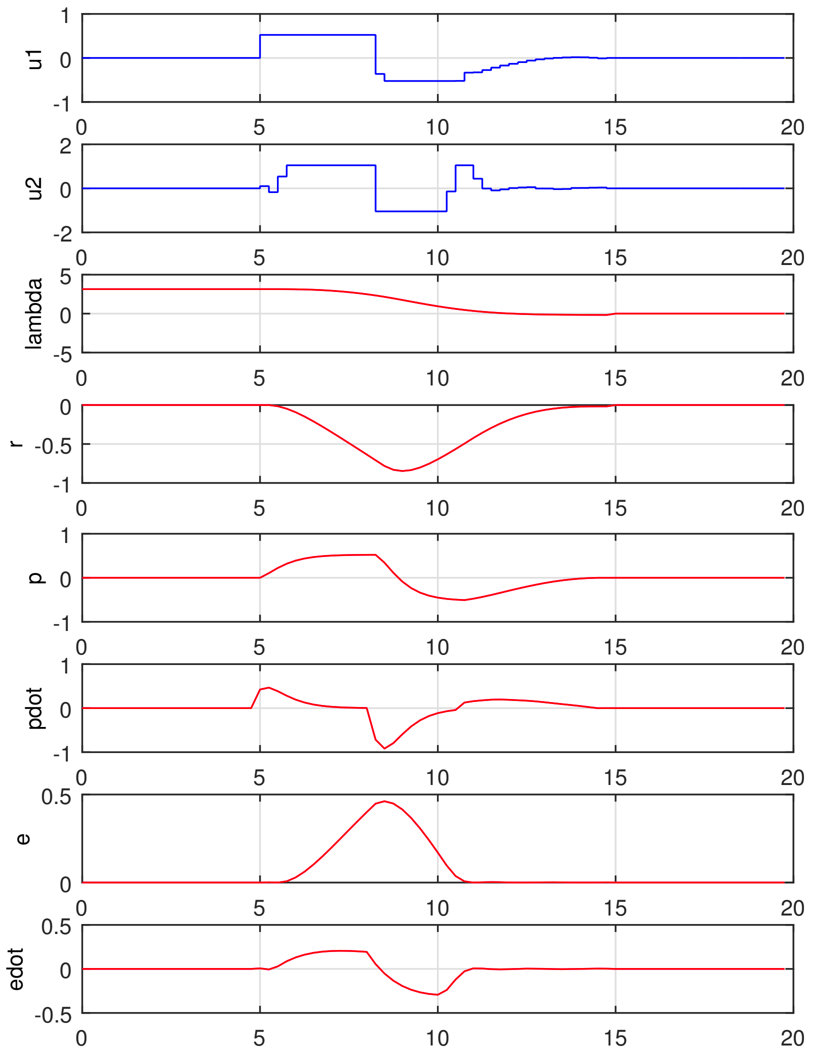

Figure 8: Optimal trajectories with elevation states and two input signals

The constraint on elevation can be described as a mountain that the helicopter have to traverse to reach its end destination. This constraint will force the system away from the linearization-point, resulting in suboptimal performance at best.

4.4 Optimal input sequence

Optimal control of pitch/travel and elevation is done with and without feedback. The feedback is introduced as defined in section 3.1 and the results are compared. The relevant LQR-weighting matrices are given bellow

$$Q_{LQR} = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} R_{LQR} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{23}$$

indicating that deviation from the optimal elevation and travel is severely punished compared to the other 4 states in the closed loop system. Constraints on the new optimal input sequence was implemented, and will be discussed further in section 4.5 and 4.6.

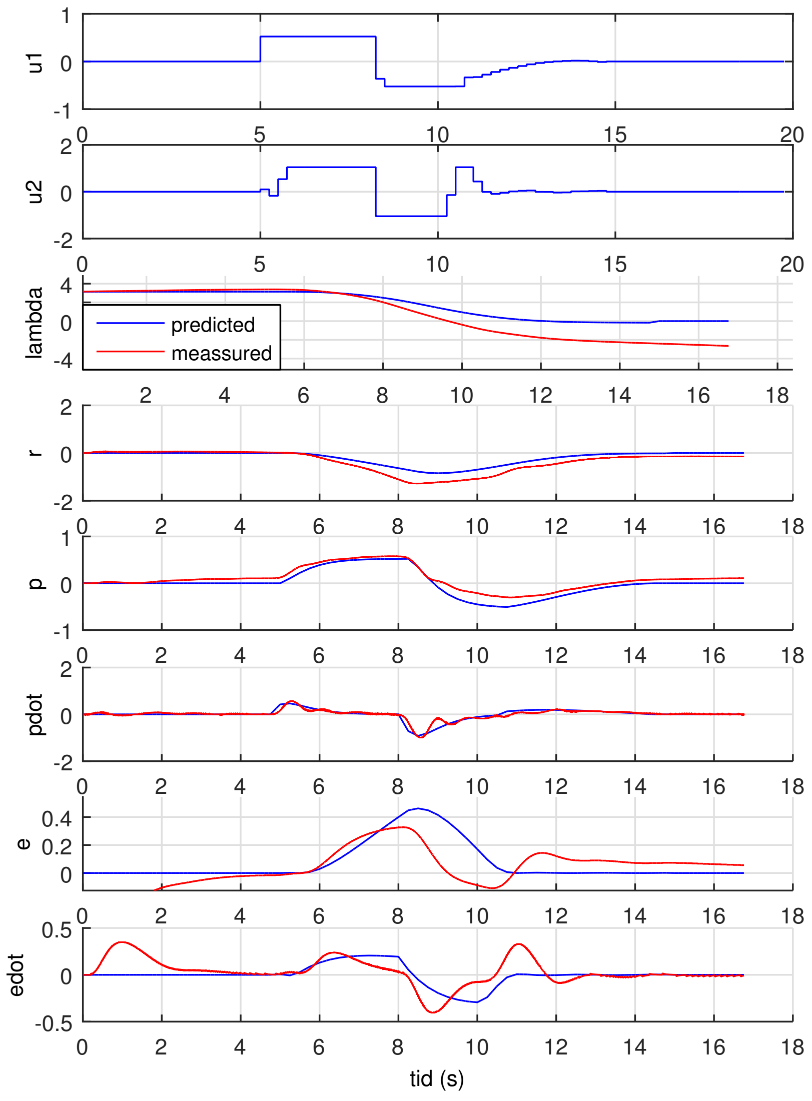

Figure 9: Measurements without feedback compared to optimal trajectories

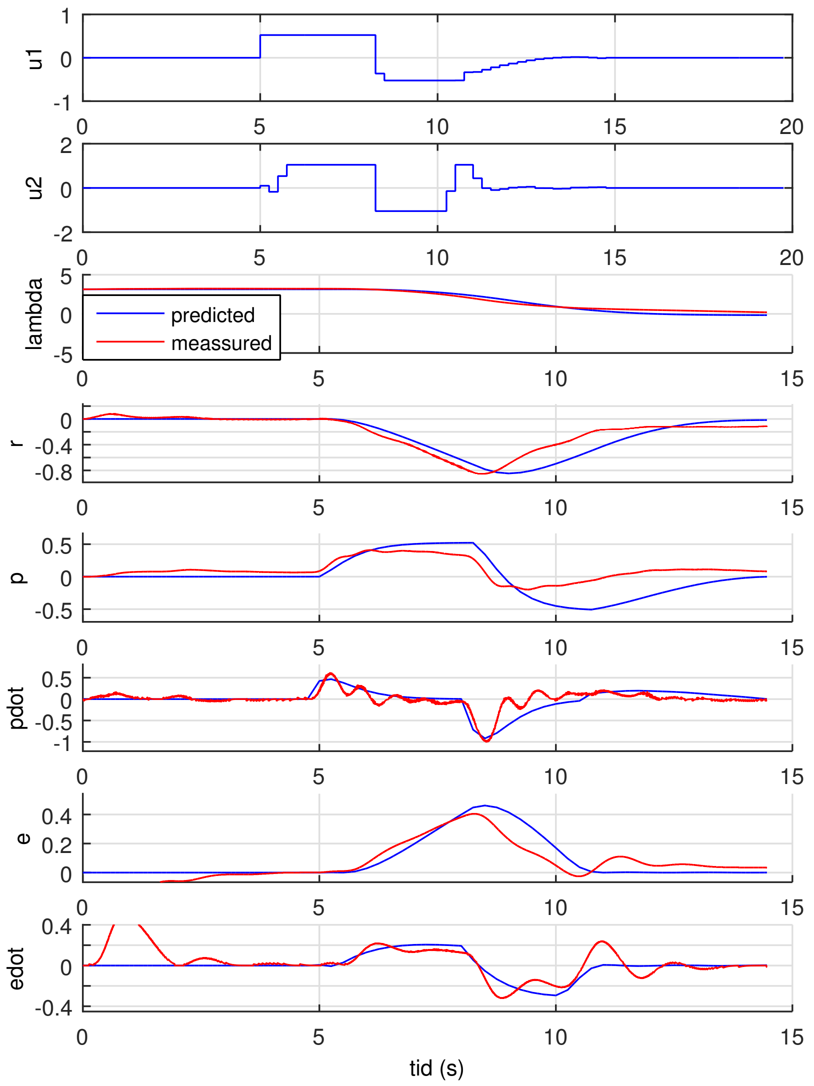

Figure 10: Measurements with feedback compared to optimal trajectories

4.5 Decoupled states

In the model, the elevation and elevation rate is decoupled from the rest of the system. This does not fit with reality, as these variables depend on the pitch-angle. The result is an optimal trajectory that does not corresponds with the physical dynamic properties of the system. A model which include this relation between the variables will result in a more accurate optimal trajectory. The solution would cause a non linear objective function, which would lead to a more complex problem, and longer solving times at each time step.

Another solution would be to tighten the upper and lower bounds on pitch. Since pitch is the shared denominator between elevation and travel, a tightening would decrease the effect of the decoupled states. This method can also be implemented on elevation as long as it does not collide with the existing constraint, creating an unfeasible set.

4.6 Discussion

Upper and lower bounds were used on pitch to minimize the effect of decoupled states. An upper and lower bound on input sequence u_1 and u_2 was implemented, the primary cause being **fmincon** [5] providing a non-reliable solution. Non-reliable as in an optimal solution showing inconsistencies with the physical limitations of the system.

In figure 9, the optimal input sequence u_k^* is used without feedback. Deviations from optimal λ and e can be observed, as expected from an open-loop system. The LQR improves this behaviour as shown in figure 10, removing the steady state deviations in λ. The measured elevation continues to lie beneath the constraint. There are several reasons for this. First, an increase in elevation increases the distance from the linerazation point, driving the physical behaviour away from the theoretical model. Secondly, there is less elevation gain when pitch is at a non-zero angle as a result of the decoupled states.

5 Discussion

Using the optimal input sequence in an open loop system resulted in expected behaviour, deviation from the optimal path was observed. Elevation and elevation rate was included in section 4. The introduction of LQR decreased deviations by implementing state feedback and penalizing λ and e, which was crucial to achieve the desired behavior. Optimal regulation of the 4 other states was not prioritized. Calculating and implementing optimal trajectories in two dimensions proved to be significantly harder, especially when the model is simplified and do not include the connection between elevation, travel and pitch.

6 Conclusion

The problem was to calculate an optimal input sequence and trajectories between two points by minimizing the quadratic cost function. The model is simplified, but performs well around the linerization-point. Constraints help improve performance in outer cases. MPC solutions comes with a high computational cost, limiting the amount of systems where an MPC is a viable solution. It should be implemented when possible.

7 Appendix

A Simulink diagrams

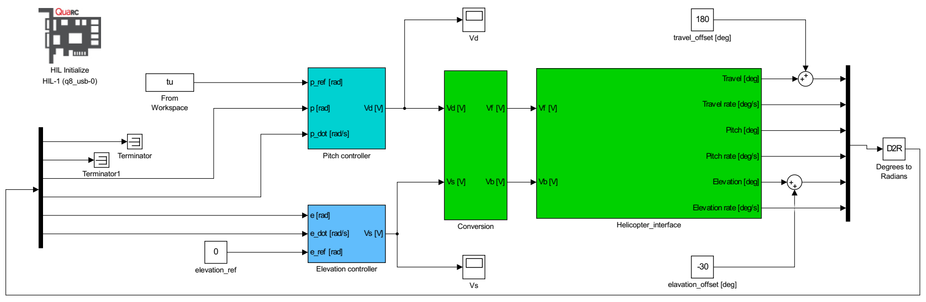

Figure 11: Simulink diagram illustrating the implementation of the system in Problem 10.2.4

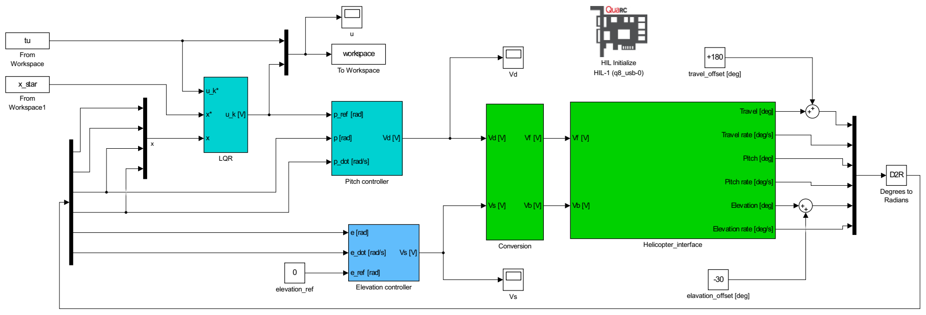

Figure 12: Simulink diagram illustrating the implementation of the system in Problem 10.3.1

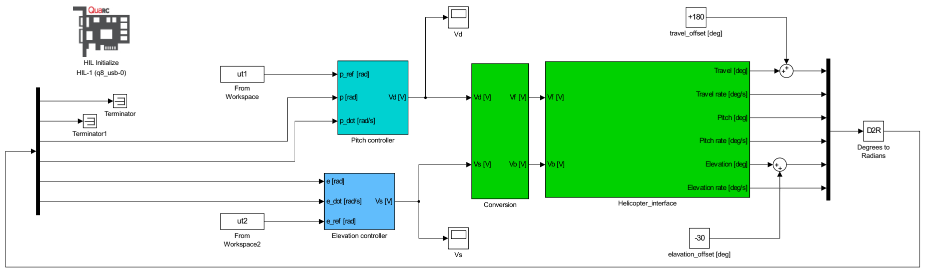

Figure 13: Simulink diagram illustrating the implementation of the system in Problem 10.4.2 without Feedback

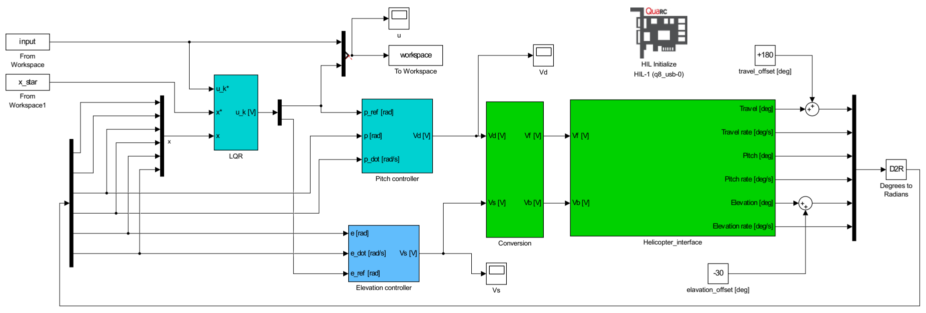

Figure 14: Simulink diagram illustrating the implementation of the system in Problem 10.4.2 with Feedback

B MATLAB scripts

B.1 Code 2.2

```matlab
% Initialization for the helicopter assignment in TTK4135.
% Run this file before you execute QuaRC -> Build.

% Updated spring 2017, Andreas L. Fl ten

init03;

%Code task 2.1
I = eye(4);
A_c = [0 1 0 0; 0 0 -K_2 0; 0 0 0 1; 0 0 -K_1*K_pp -K_1*K_pd];
B_c = [0; 0; 0; K_1*K_pp];

%Code task 2.2
Theta_t = 0.25;
A = I + Theta_t*A_c;
B = Theta_t*B_c;
```

B.2 Code 2.3

```
% TTK4135 - Helicopter lab
% Hints/template for problem 2.
% Updated spring 2017, Andreas L. Fl?ten

%% Initialization and model definition
code22;
delta_t = 0.25; % sampling time

% Discrete time system model. x = [lambda r p p_dot]'
A1 = A;
B1 = B;

% 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]         = hints_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 (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 = 2*hints_genq2(Q1,P1,N,M,mu);% Generate Q


%% Generate system matrixes for linear model
Aeq = hints_gena2(A1,B1,N,mx,mu);% Generate A
beq = zeros(mx*N, 1); % Generate b
beq(1:mx) = A1*x0;    % Initial value

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

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(23)
subplot(511)
stairs(t,u),grid
ylabel('u')
subplot(512)
plot(t,x1,'m',t,x1,'r'),grid
ylabel('lambda')
subplot(513)
plot(t,x2,'m',t,x2','r'),grid
ylabel('r')
subplot(514)
plot(t,x3,'m',t,x3,'r'),grid
ylabel('p')
subplot(515)
plot(t,x4,'m',t,x4','r'),grid
xlabel('tid (s)'),ylabel('pdot')
hold off

ut = [t' u];
```

B.3 Code 2.4

```
% TTK4135 - Helicopter lab
% Hints/template for problem 2.
% Updated spring 2017, Andreas L. Fl?ten

%% Initialization and model definition
code23; %

%% Plotting
load('24.mat');

figure(24)
subplot(511)
stairs(t,u),grid
ylabel('u')
subplot(512)
hold on
lambda_prediction = plot(x_state(1,:),x_state(2,:),'b');
lambda_meassured = plot(x_state(1,:),x_state(6,:),'r');
legend([lambda_prediction, lambda_meassured], 'predicted'
    , 'meassured');
hold off
ylabel('lambda')
subplot(513)
hold on
r_prediction = plot(x_state(1,:),x_state(3,:),'b');
r_meassured = plot(x_state(1,:),x_state(7,:),'r');

hold off
ylabel('r')
subplot(514)
hold on
pitch_prediction = plot(x_state(1,:),x_state(4,:),'b');
pitch_meassured = plot(x_state(1,:),x_state(8,:),'r');

hold off
ylabel('p')
subplot(515)
hold on
pitch_rate_prediction = plot(x_state(1,:),x_state(5,:),'b
    ');
pitch_rate_meassured = plot(x_state(1,:),x_state(9,:),'r'
    );

hold off
```

```
xlabel('tid␣(s)'),ylabel('pdot')

%For simulink
x_star = [t' x1 x2 x3 x4];
ut = [t' u];
```

B.4 Code 3.1

```
% TTK4135 - Helicopter lab
% Hints/template for problem 2.
% Updated spring 2017, Andreas L. Fl?ten

%% Initialization and model definition
code22;
delta_t = 0.25; % sampling time

% Discrete time system model. x = [lambda r p p_dot]'
A1 = A;
B1 = B;

% 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]       = hints_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) = 0;                         % Weight on state x3
Q1(4,4) = 0;                         % Weight on state x4
P1 = 1;                              % Weight on input
Q = 2*hints_genq2(Q1,P1,N,M,mu);% Generate Q

%% Generate system matrixes for linear model
Aeq = hints_gena2(A1,B1,N,mx,mu);   % Generate A
beq = zeros(mx*N, 1);               % Generate b
beq(1:mx) = A1*x0;                  % Initial value

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

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);
tu = [t' u];


%% LQR
Q_LQR = [2 0 0 0; 0 1 0 0; 0 0 1 0; 0 0 0 1];
R_LQR = [0.1];
K = dlqr(A, B, Q_LQR, R_LQR)
x_star = [t', x1, x2, x3, x4];

figure(31)
load('31.mat')
plot(workspace.time, workspace.signals.values)
xlabel('tid␣(s)'),ylabel('u_k*,␣u_k')
```

B.5 Code 3.2

```
% TTK4135 - Helicopter lab
% Hints/template for problem 2.
% Updated spring 2017, Andreas L. Fl?ten

%% Initialization and model definition
code31;

%% Plotting

load('32.mat');

figure(32)
subplot(511)
stairs(t,u),grid
ylabel('u')
subplot(512)
hold on
lambda_prediction = plot(x_state(1,:),x_state(2,:),'b');
lambda_meassured = plot(x_state(1,:),x_state(6,:),'r');
legend([lambda_prediction, lambda_meassured], 'predicted'
    , 'meassured');
hold off
ylabel('lambda')
subplot(513)
hold on
r_prediction = plot(x_state(1,:),x_state(3,:),'b');
r_meassured = plot(x_state(1,:),x_state(7,:),'r');
hold off
ylabel('r')
subplot(514)
hold on
p_prediction = plot(x_state(1,:),x_state(4,:),'b');
p_meassured = plot(x_state(1,:),x_state(8,:),'r');
hold off
ylabel('p')
subplot(515)
hold on
p_rate_prediction = plot(x_state(1,:),x_state(5,:),'b');
p_rate_meassured = plot(x_state(1,:),x_state(9,:),'r');
hold off
xlabel('tid␣(s)'),ylabel('pdot')

%% For simulink
x_star = [t', x1, x2, x3, x4];
```

```
tu = [t' u];
```

B.6 Code 4.2

```
% Initialization for the helicopter assignment in TTK4135
    .
% Run this file before you execute QuaRC -> Build.

% Updated spring 2017, Andreas L. F l ten

init03;

%Code task 4.1
I = eye(6);
A_c = [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];
B_c = [0 0; 0 0; 0 0; K_1*K_pp 0; 0 0; 0 K_3*K_ep];

%Code task 4.2
Theta_t = 0.25;
A = I + Theta_t*A_c;
B = Theta_t*B_c;
```

B.7 Code 4.3

```
% TTK4135 - Helicopter lab
% Hints/template for problem 2.
% Updated spring 2017, Andreas L. Fl?ten

%% Initialization and model definition
code42;

global N mx

% 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
x0_lambda   = pi;
x0_r        = 0;
x0_p        = 0;
x0_p_dot    = 0 ;
x0_e        = 0;
x0_e_dot    = 0;
x0 = [x0_lambda x0_r x0_p x0_p_dot x0_e x0_e_dot]';

% Time horizon and initialization
N  = 40;                 % Time horizon task 4
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

% Generate the matrix Q and the vector c (objecitve
    function weights in the QP problem)
q1 = 1;
q2 = 1;

%% Generate system matrixes for linear model
Aeq = hints_gena2(A,B,N,mx,mu);
beq = [A*x0; zeros((N-1)*mx,1)];

%Bounds
lowerBounds = [
     repmat([-Inf; -Inf; -Inf; -Inf; -Inf; -Inf], N, 1);
     repmat([-pi/6; -pi/3], N, 1)];
```

```

upperBounds = [
     repmat([Inf; Inf; Inf; Inf; Inf; Inf], N, 1);
     repmat([pi/6; pi/3], N, 1)];

%% Solve QP problem with linear model
% objective function
myf = @(lambda, p, e) ones(1, N)*(lambda.^2 + q1.*p.^2 +
    q2.*e.^2);
f = @(x) myf(x(1: 6: 6*N), x(3 : 6: 6*N), x(5: 6: 6*N));

options = optimset('Algorithm','active-set');
options = optimset(options, 'MaxFunEvals', 60000);
tic
[z, fval] = fmincon(f,z0,[],[],Aeq,beq,lowerBounds,
    upperBounds,@func_constraint, options);
t1=toc;

%% Extract control inputs and states
u1 = z(N*6 + 1:2:N*8);
u2 = z(N*6 + 2:2:N*8);

x1 = z(1:6:N*6);
x2 = z(2:6:N*6);
x3 = z(3:6:N*6);
x4 = z(4:6:N*6);
x5 = z(5:6:N*6);
x6 = z(6:6:N*6);

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

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

%% Plotting
t = 0:Theta_t:Theta_t*(length(u1)-1);

figure(43)
```

```
subplot(811)
stairs(t,u1),grid
ylabel('u1')
subplot(812)
stairs(t,u2),grid
ylabel('u2')
subplot(813)
plot(t,x1,'m',t,x1,'r'),grid
ylabel('lambda')
subplot(814)
plot(t,x2,'m',t,x2','r'),grid
ylabel('r')
subplot(815)
plot(t,x3,'m',t,x3,'r'),grid
ylabel('p')
subplot(816)
plot(t,x4,'m',t,x4','r'),grid
ylabel('pdot')
subplot(817)
plot(t,x5,'m',t,x5','r'),grid
ylabel('e')
subplot(818)
plot(t,x6,'m',t,x6','r'),grid
ylabel('edot')
hold off

input = [t' u1 u2];
x_star = [t' x1 x2 x3 x4 x5 x6];

ut1 = [t' u1];
ut2 = [t' u2];
```

B.8 Code 4.4

```
% TTK4135 - Helicopter lab
% Hints/template for problem 2.
% Updated spring 2017, Andreas L. Fl?ten

%% Initialization and model definition
code43;

%% LQR
Q_LQR = [3 0 0 0 0 0; 0 1 0 0 0 0; 0 0 1 0 0 0; 0 0 0 1 0
    0; 0 0 0 0 20 0; 0 0 0 0 0 1];
R_LQR = [1 0; 0 1];
K = dlqr(A, B, Q_LQR, R_LQR);

%% Plotting
load('442.mat');

figure(442)
subplot(811)
stairs(t,u1),grid
ylabel('u1')
subplot(812)
stairs(t,u2),grid
ylabel('u2')
subplot(813)
hold on
lambda_prediction = plot(x_state(1,:),x_state(2,:),'b'),
    grid;
lambda_meassured = plot(x_state(1,:),x_state(8,:),'r');
legend([lambda_prediction, lambda_meassured], 'predicted'
    , 'meassured');
hold off
ylabel('lambda')
subplot(814)
hold on
plot(x_state(1,:),x_state(3,:),'b'), grid;
plot(x_state(1,:),x_state(9,:),'r');
hold off
ylabel('r')
subplot(815)
hold on
plot(x_state(1,:),x_state(4,:),'b'), grid;
plot(x_state(1,:),x_state(10,:),'r');
hold off
ylabel('p')
```

```
subplot(816)
hold on
plot(x_state(1,:),x_state(5,:),'b'), grid;
plot(x_state(1,:),x_state(11,:),'r');
hold off
ylabel('pdot')
subplot(817)
hold on
plot(x_state(1,:),x_state(6,:),'b'), grid;
plot(x_state(1,:),x_state(12,:),'r');
hold off
ylabel('e')
subplot(818)
hold on
plot(x_state(1,:),x_state(7,:),'b'), grid;
plot(x_state(1,:),x_state(13,:),'r');
hold off
ylabel('edot')
xlabel('tid␣(s)')

%%For simulink
input = [t' u1 u2];
x_star = [t' x1 x2 x3 x4 x5 x6];
```

B.9 Constraint function

```
function [ c, ceq ] = func_constraint( z )
%FUNC_CONSTRAINT Summary of this function goes here
%   Detailed explanation goes here
global N mx

alpha = 0.5;
beta = 20;
lambda_t = (2*pi)/3;

c_temp = -Inf(N,1);
lambda_i = z(1:mx:N*mx);
ek = z(5:mx:N*mx);

for n = 1:N
    c_temp(n) = alpha*exp(-beta*(lambda_i(n)-lambda_t)^2)
         - ek(n);
end
ceq = [];
c = c_temp;

end
```

References

[1] Assignment text, *Department of Engineering Cybernetics*, NTNU, Trondheim, February, 2017.

[2] Discretization, `https://en.wikipedia.org/wiki/Discretization`, From Wikipedia, the free encyclopedia, updated 24 March 2017

[3] Euler method, `https://en.wikipedia.org/wiki/Euler_method`, From Wikipedia, the free encyclopedia, updated 4 April 2017

[4] Merging Optimization and Control, *Bjarne Foss and Tor Aksel N. Heirung Department of Engineering Cybernetics*, NTNU, Trondheim,

[5] Nonlinear programming solver, `https://se.mathworks.com/help/optim/ug/fmincon.html?s_tid=gn_loc_drop`, From MathWorks, updated 4 April 2017

[6] Linear-quadratic (LQ) state-feedback regulator for discrete-time state-space system, `https://se.mathworks.com/help/control/ref/dlqr.html?requestedDomain=www.mathworks.com`, From MathWorks, updated 21 April 2017