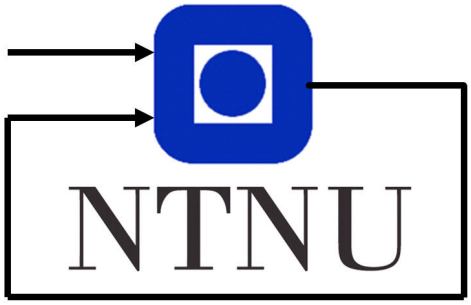
TTK4135 Optimisation and Control
Helicopter Project

Group 5
Student 528070
Student 528146

April, 2022



Department of Engineering Cybernetics

Contents

1 10.2 - Optimal Control of Pitch/Travel without Feedback

In this part of the project, the travel of the helicopter is controlled towards a reference. This is done through a pitch reference output by a quadratic optimisation problem.

1.1 The continuous model

A continuous model for the helicopters travel and pitch is given as a set of linear equations in section 8 of the lab handout [1]:

$$\dot{\lambda} = r \tag{1a}$$
$$\dot{r} = -K_2 p \tag{1b}$$
$$\dot{p} = \dot{p} \tag{1c}$$
$$\ddot{p} = -K_1 K_{pd}\dot{p} - K_1 K_{pp} p + K_1 K_{pp} p_c \tag{1d}$$

where the variables and parameters are defined in tables 1 and 2 of the lab handout[1]. Defining the state vector $x = [\lambda \;\; r \;\; p \;\; \dot{p}]^\top$ and the input $u = p_c$, this can readily be expressed as the following state space model:

$$\dot{x} = A_c x + B_c u \tag{2}$$

where

$$A_c = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1 K_{pp} \end{bmatrix}.$$

(2) models the dynamics of the helicopters travel, travel rate, pitch and pitch rate when a reference p_c is given to a PD controller for pitch. In this case, the pitch reference serves as the "input" to the model and the state vector x serves as the output. Figure 1 shows the overview of the entire system in this part of the project, modified from figure 7 of the lab handout [1]. A dotted line has been added around the parts modelled by (2). Note that the input to our model, $u = p_c$, is the output of the "optimisation layer". Because the PD controller is embedded in the model, (2) models both the physical plant dynamics, from motor input voltage to state dynamics, as well as the "basic control layer", from a pitch reference to a motor voltage. The motor input is implicit in the model in the sense that it has been replaced by a PD control law, and its relationship with the state dynamics is given through the constant K_1. This is a useful abstraction because the basic control layer is not a focus in this project. Rather, we focus on how the optimisation layer affects the state x, and this focus is reflected by the choice of model.

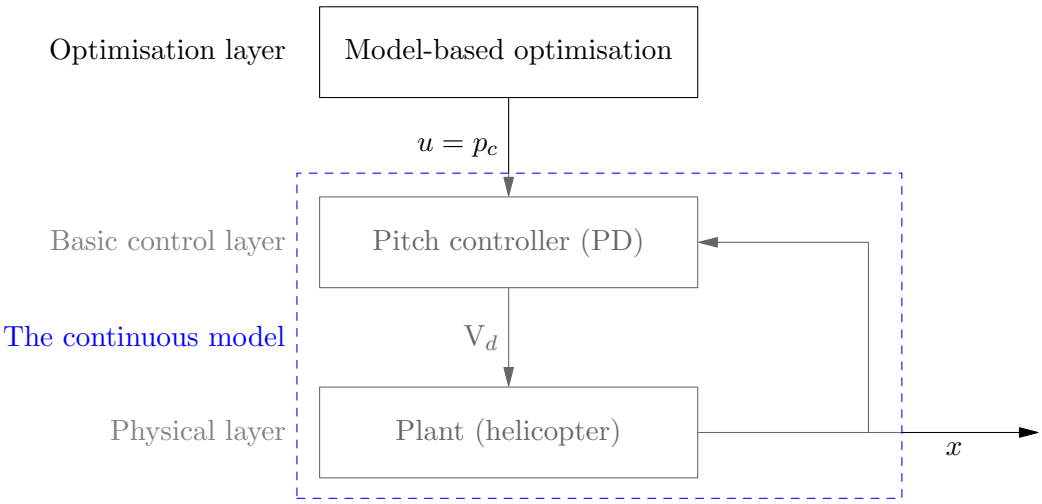


Figure 1: Modified illustration of control hierarchy, from lab handout [1].

To make a note on accuracy, the PD controller is implemented as a discrete controller on the computer, and can be assumed to correctly describe the controller dynamics. Any uncertainties in the model can therefore be attributed to the physical plant dynamics.

1.2 The discretised model

The discretisation of (2) is performed using forward Euler discretisation, which uses a simple linear approximation given by

$$\frac{x(t+\Delta t) - x(t)}{\Delta t} \approx \dot{x} = A_c x(t) + B_c u(t). \tag{3}$$

where Δt denotes the step time (see section 1.7). Defining $x_k := x(k\Delta t)$ and $u_k := u(k\Delta t)$ for $k = 0, 1, 2, \ldots$, the forward Euler discretisation becomes

$$x_{k+1} = Ax_k + Bu_k \tag{4}$$

where

$$A = I + A_c\Delta t = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & -K_2\Delta t & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & -K_1K_{pp}\Delta t & 1 - K_1K_{pd}\Delta t \end{bmatrix},$$

$$B = B_c\Delta t = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1K_{pp}\Delta t \end{bmatrix}.$$

1.3 The open loop optimisation problem

In open loop optimisation, a cost function of the state and input is minimised over a time horizon, and the minimising input sequence is implemented. The cost function selected for the problem of controlling travel is given by:

$$\Phi = \sum_{i=0}^{N-1} (\lambda_{i+1} - \lambda_f)^2 + q p_{ci}^2, \quad q \geq 0 \tag{5}$$

where N is the time horison, λ_{i+1} is the travel at time step $i+1$, λ_f is the reference travel, p_{ci} is the pitch reference at timestep i and q is a nonegative weight.

Briefly put, minimising the first term in (5) is desirable because it provides a trajectory where the travel is close to the reference for most of the time horizon. The second term is included such that the control trajectory is kept to reasonable control input sizes. Obviously, minimising (5) alone is not enough as this does not force the trajectory to adhere to the model developed in section 1.2. This model imposes a linear constraint on the states and inputs of the optimal trajectory. In addition, the project task [1] imposes that the pitch reference and the pitch stay within ± 30 degrees, which are simple inequality constraints on the input and state:

$$\begin{aligned} -30^\circ \leq p_c &\leq 30^\circ \\ -30^\circ \leq p &\leq 30^\circ \end{aligned}$$

The quadratic nature of (5) and linearity of the constraint makes this a quadratic programming problem (QP). In order to solve the QP outlined above using standard optimisation software, it must first be rephrased in terms of matrix operations. The representation chosen for this project is the so-called full space representation [2] of the problem. We begin by defining the vector

$$z := [x_1, \cdots, x_N, u_0, \cdots, u_{N-1}]^T. \tag{6}$$

i.e. the vector containing all states and inputs over the horizon N and where $u_i = p_{ci}$.

In terms of this new state-input vector, minimising the objective function (5) can be formulated as minimising a standard quadratic:

$$\min_z \frac{1}{2} z^\top G z + c^\top z, \tag{7}$$

where the linear term $c^\top z$ arises due to λ_f in our case. Note that the term λ_f^2 in (5) is constant, and hence will not impact the result of the minimisation.

In this project, the travel reference is $\lambda_f = 0$, such that also the linear term disappears. Furthermore, a constant $1/2$ is included because this is the way Matlabs quadratic programming solver expects its input. The resulting optimisation problem is then given by:

$$\min_z \frac{1}{2} z^T \begin{bmatrix} 2Q & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & 2Q & \ddots & & \vdots \\ \vdots & & \ddots & 2q & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 2q \end{bmatrix} z, \tag{8a}$$

subject to the constraints

$$x_{t+1} = Ax_t + Bu_t, \qquad t = 0, \cdots, N-1 \tag{8b}$$

$$\begin{bmatrix} -\infty \\ -\infty \\ -30^\circ \\ -\infty \end{bmatrix} \leq x_t \leq \begin{bmatrix} \infty \\ \infty \\ 30^\circ \\ \infty \end{bmatrix}, \qquad t = 1, \cdots, N \tag{8c}$$

$$-30^\circ \leq u_t \leq 30^\circ, \qquad t = 0, \cdots, N-1 \tag{8d}$$

where x_0 is given and Q is given by:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The constraints from (8) must for the implementation also be rephrased into matrix operations on the state-input vector z. The constant constraints (8c) and (8d) can simply be repeated into long column vectors to fit the dimension of z. The model constraint (8b) was rephrased like this:

$$A_{eq} z = b_{eq} \tag{9}$$

with

$$A_{eq} = \begin{bmatrix} I & 0 & \cdots & \cdots & 0 & -B & 0 & \cdots & \cdots & 0 \\ -A & \ddots & & & \vdots & 0 & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 & \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -A & I & 0 & \cdots & \cdots & 0 & -B \end{bmatrix}, b_{eq} = \begin{bmatrix} Ax_0 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}.$$

The implementation of the optimisation problem is shown in section 1.7, where the matrices (9) were generated using the provided function `gen_aeq`.

1.4 The weights of the optimisation problem

As is seen from the objective function (5), the only variable is the weight q, which linearly scales the square of the input, p_{ci}^2. Since the quadratic program (8) minimises the objective function, changing the scale of *only* the input term will lead to a different optimal state and input trajectory that takes the scaling into account. Practically, a higher input weight q will result in a control sequence with smaller values, and hence a slower state convergence because the objective "punishes" large inputs more than large state deviations. On the contrary, a smaller q will minimise state deviations more than inputs, which leads to faster convergence at the expense of larger inputs. This general behaviour is confirmed by the solutions to the optimisation problem for different values of q, plotted in Figure 2. Notice also how the input saturates at $\pm$ 30 degrees due to the constraints on the optimisation problem, and how the solution spends more time at these maxima for lower q weights.

1.5 The objective function

The objective function (5) and the corresponding open loop optimisation problem (8) give a control solution tailored to the theoretical model. Minimising the square of the relative travel, $(\lambda - \lambda_f)^2$, naturally leads to a trajectory where the travel is as close as possible to the target λ_f. Furthermore, the quadratic nature of the term means that the controller will produce larger inputs for larger deviations. In practice, this means large initial inputs that bring the state to the vicinity of the reference, and then gradually smaller inputs to fully converge to it. Another effect of the quadratic is that only the *size* of the deviations, and not their *sign*, matters.

Although it is desirable for large state deviations to be corrected more aggressively than small deviations, a possible disadvantage of the quadratic objective function could be that, as the travel approaches its reference, the state-weighing-term $(\lambda - \lambda_f)^2$ in the objective becomes very small. This could be problematic because the control input only deals with small offsets very slowly, and might even never reach the reference fully. This could be remedied by incorporating a linear term weighing the absolute value of the state deviation.

Another interesting consideration with the objective function is that it does not take in account that the helicopter always can approach its reference travel λ_f in a clockwise and counterclockwise direction. This is illustrated in Figure 3. If the initial position is at $\lambda = 270^\circ$ and we set $\lambda_f = 0^\circ$, the optimisation algorithm will give the red path shown in the figure, and not the obviously shorter green path. This is a matter of how we define

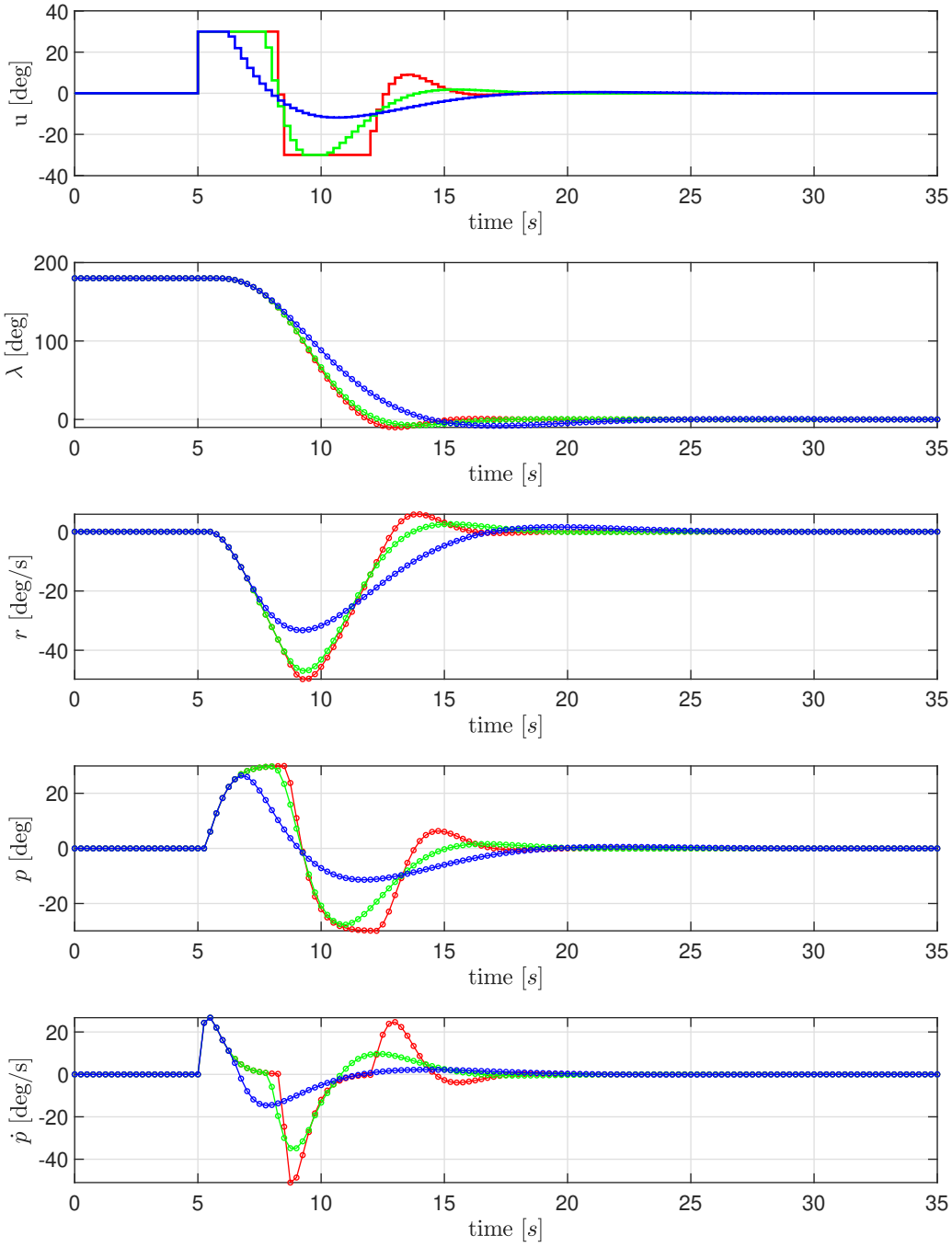


Figure 2:
Calculated optimal state and input trajectories for
● $q = 0.1$ ● $q = 1$ ● $q = 10$

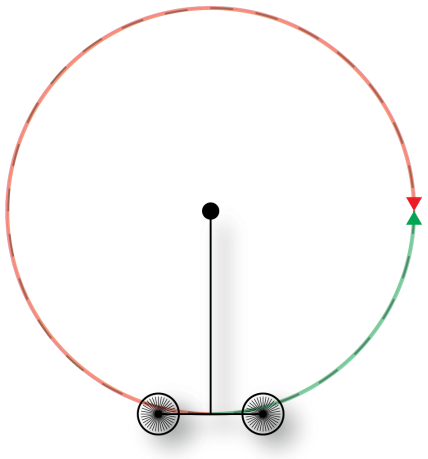


Figure 3: Birdseye view of helicopter, with two possible paths, in green and red, to a desired position.

angles, and whether 0° should equal 360°. If the only requirement is to reach the point on the circle characterised by the reference travel, relative angle definitions should be introduced.

1.6 Experimental results

Figure 4 displays the experimental results when the helicopter was provided the open-loop optimal input sequence from the optimisation problem (8) with $q = 1$. The experimental results drastically differ to the simulated trajectory of Figure 2. Specifically, though the pitch and pitch rate follow the reference relatively well, the travel tends to grow quite significantly and shows no signs of stopping, as is seen from the final nonzero travel rate! The reason for this is that, from the travels' perspective, the control applied is completely open-loop. Because the optimisation calculates an optimal trajectory purely from the initial state and reference, it does not take into account the state measurements during the execution. This has the the same downside as any open-loop control system, namely that it does not handle disturbances or model inaccuracies. In our case, any minor model inaccuracy or disturbance in pitch and travel will affect the travel directly, without correction, and hence cause the travel to differ from the optimal trajectory. The pitch does not suffer from this because it is affected by a closed loop PD-controller that ensures it follows the reference.

To investigate the travel deviation further, the helicopter was run with a constant zero pitch reference, and a counterclockwise rotation in travel was observed each time. Due to the uniformity of the travel error across trials, we hypothesise the main modelling error causing the deviation to be that both blades on the helicopter are mounted the same way. This means that running both at the same voltage will generate a slight torque about the travel axis, causing the helicopters travel to be affected even when the pitch is zero.

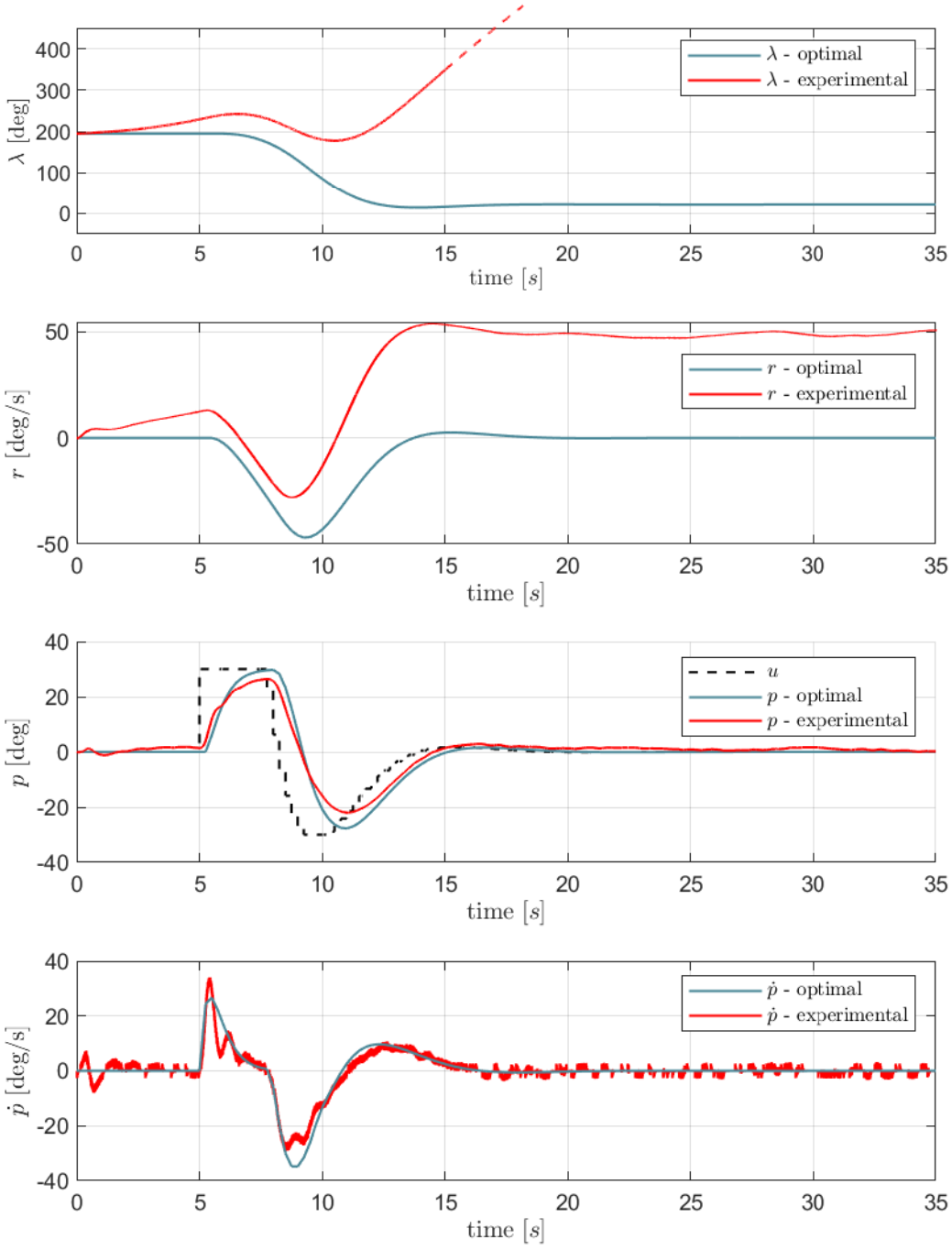


Figure 4: Experimental and optimal trajectories with $q = 1$.

1.7 MATLAB and Simulink

Listing 1: The matlab script for task 2.

```
%% Initialization and model definition
init05;
% Discrete time system model. x = [lambda r p p_dot]'
delta_t = 0.25; % sampling time
A = [1 delta_t 0 0;
     0 1 -delta_t*K_2 0;
     0 0 1 delta_t;
     0 0 -K_1*K_pp*delta_t 1-K_1*K_pd*delta_t];
B = [0; 0; 0; K_1*K_pp*delta_t];

% 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
x0 = [x1_0 x2_0 x3_0 x4_0]';            % Initial values

% Time horizon and initialization
N  = 100;                               % Time horizon for states
M  = N;                                 % Time horizon for inputs
z  = zeros(N*mx+M*mu,1);                % Initialize z for the whole horizon
z0 = z;                                 % Initial value for optimization

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

xl      = -Inf*ones(mx,1);              % Lower bound on states (no bound)
xu      = Inf*ones(mx,1);               % Upper bound on states (no bound)
xl(3)   = ul;                           % Lower bound on state x3
xu(3)   = uu;                           % Upper bound on state x3

% Generate constraints on measurements and inputs
[vlb,vub]       = gen_constraints(N,M,xl,xu,ul,uu);
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)
Q = zeros(mx,mx);
Q(1,1) = 1;                             % Weight on state x1
Q(2,2) = 0;                             % Weight on state x2
Q(3,3) = 0;                             % Weight on state x3
Q(4,4) = 0;                             % Weight on state x4
q = 1;                                  % Weight on input
G = gen_q(2*Q,2*P,N,M);
c = zeros(1,length(z));                 % Generate c, this is the linear constant
     term in the QP

%% Generate system matrixes for linear model
Aeq = gen_aeq(A,B,N,mx,mu);
beq = zeros(N*length(x0),1);
beq(1:4) = A*x0;

```

```
57
58 %% Solve QP problem with linear model
59 tic
60 [z,lambda] = quadprog(G,c,[],[],Aeq,beq,vlb,vub,x0);
61 t1=toc;
62
63 % Calculate objective value
64 phi1 = 0.0;
65 PhiOut = zeros(N*mx+M*mu,1);
66 for i=1:N*mx+M*mu
67   phi1=phi1+Q(i,i)*z(i)*z(i);
68   PhiOut(i) = phi1;
69 end
70
71 %% Extract control inputs and states
72 u  = [z(N*mx+1:N*mx+M*mu);z(N*mx+M*mu)]; % Control input from solution
73
74 x1 = [x0(1);z(1:mx:N*mx)];              % State x1 from solution
75 x2 = [x0(2);z(2:mx:N*mx)];              % State x2 from solution
76 x3 = [x0(3);z(3:mx:N*mx)];              % State x3 from solution
77 x4 = [x0(4);z(4:mx:N*mx)];              % State x4 from solution
78
79 num_variables = 5/delta_t;
80 zero_padding = zeros(num_variables,1);
81 unit_padding  = ones(num_variables,1);
82
83 u   = [zero_padding; u; zero_padding];
84 x1  = [pi*unit_padding; x1; zero_padding];
85 x2  = [zero_padding; x2; zero_padding];
86 x3  = [zero_padding; x3; zero_padding];
87 x4  = [zero_padding; x4; zero_padding];
88 %% Time series input
89 time_steps = [0:delta_t:delta_t*(length(u)-1)]';
90 u_t = timeseries(u,time_steps);
```

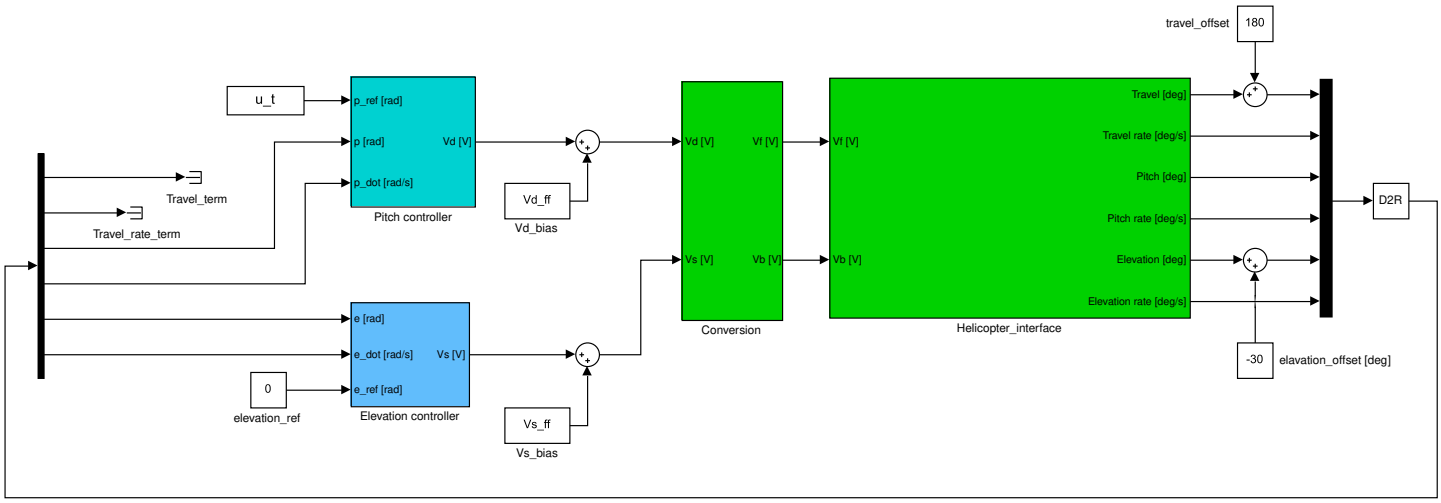


Figure 5: The simulink diagram for task 2.

2 10.3 - Optimal Control of Pitch/Travel with Feedback (LQ)

In this part of the project, feedback is introduced by steering the helicopter towards an optimal trajectory using an LQ controller.

2.1 LQ controller

The linear quadratic (LQ) controller is a linear state feedback control law designed to bring a system towards zero state. This can be used to bring the system states towards the trajectory output by the open loop optimisation problem (8) by defining it as a state *deviation* feedback, given by the following control law, evaluated in each time step:

$$u = u^* - K^T(x - x^*), \tag{10}$$

where K is a gain matrix and $\{u^*, x^*\}$ denotes the optimal trajectory as given from (8). This effectively transforms the open loop optimisation control from section 1 into a closed loop control law that takes into accounts disturbances and modelling errors, which most certainly are present as discussed in section 1.6.

The gain matrix K is itself calculated by minimising the following cost function of the state and input:

$$J = \sum_{i=0}^{\infty} \Delta x_{i+1}^\top Q_{LQR} \Delta x_{k+i} + \Delta u_i^\top R_{LQR} \Delta u_i, \quad Q_{LQR} \geq 0, R_{LQR} > 0 \tag{11}$$

where

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

Here, the squares of state and input deviations from the optimal trajectory are weighed through the weighing matrices Q_{LQR} and R_{LQR}. The calculation of K was done with the Matlab function *dlqr*, and the corresponding code is provided in section 2.4. The ratio between the elements in the weighing matrices determines the relative importance assigned to minimising the different state deviations from the optimal as well as the input deviation from the optimal. This is highly analogous to the way the open loop objective function (5) uses weights in section 1.5, and was thoroughly discussed in the course TTK4115 linear systems theory. A selection of weighing matrices are experimentally investigated and motivated in section 2.3.

2.2 Model Predictive Control

An alternative to the state feedback LQ controller realised in this part of the project, is model predictive control, or MPC. MPC involves solving an

entirely new finite horizon open loop optimisation problem at each time step, initialised with the current state as initial conditions. The control applied to the system at each time step is simply the first control value output by the optimisation problem. This control method could have been implemented in Simulink by feeding the state and input into a Matlab function that runs the optimisation problem and outputs the optimal trajectory. The first input from this trajectory is used, and the process is repeated next time step. The hierarchical control structure is presented in Figure 6.

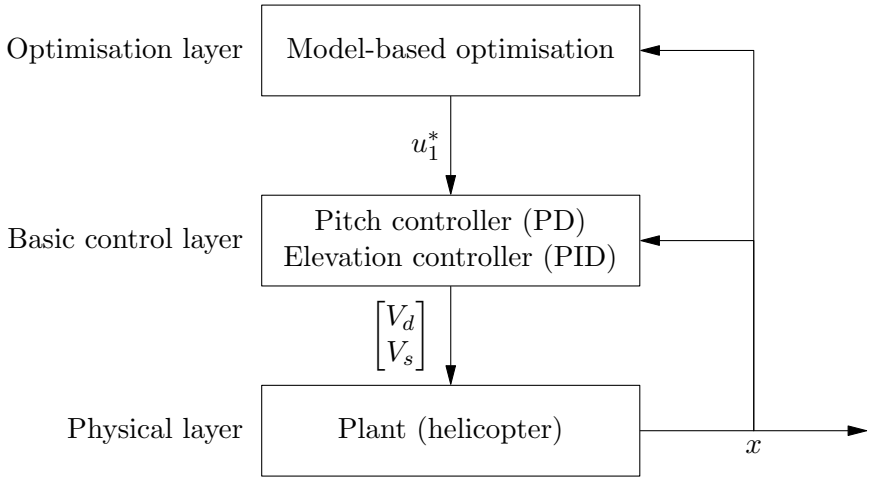


Figure 6: Illustration of the layers in the control hierarchy when using a MPC controller. Only the first element from the optimal input sequence is fed to the basic control layer, and the state is fed back to the optimisation layer as an initial condition.

Though it initially is not evident, this is as much a closed loop control mechanism as its LQ counterpart. The closing of the loop is done more subtly by feeding the current state back as the initial conditions to the new optimisation problem. Hence, any disturbances or modelling inaccuracies are accounted for by the new trajectory through this initial condition. An advantage of MPC over LQ control is that it in many ways is more *optimal*. Which trajectory is actually the best to take, provided a specific set of weighing matrices, is strongly influenced by where the system is. For the MPC, this is no issue as the optimal trajectory is recomputed at each time step to account for possible deviations from the initially computed trajectory. The LQ controller however always pushes the system towards the trajectory that *was* optimal from the initial state, but that very well may be sub-optimal if the state has deviated due to disturbances or other errors.

A further advantage of MPC over LQ control is that it will never produce

an input or, for our application, a pitch reference that violates the input constraints. In contrast, the LQ controller provides no such guarantee, as the output control input will increase with deviation from the optimal trajectory. This is further discussed in section 2.3. In a system where it is vital to satisfy input constraints, MPC may therefore be preferable to LQ control.

However, an argument can be made for the LQ controller in regards to efficiency, especially if it is run on an embedded, computationally weak system. This is simply because computing the linear state feedback in (10) is a significantly smaller operation than computing an entire optimisation problem at each time step.

2.3 Experimental results

Table 1: Tested configurations of Q_{LQR} and R_{LQR} and corresponding figures

$\mathbf{diag(Q_{LQR})}$	$\mathbf{R_{LQR}}$	**Figure**
1 1 1 1	1	Figure 7
100 1 1 1	1	Figure 8
1 1 1 1	1000	Figure 9
1000 10 1 100	1	Figure 10

The open loop optimisation problem in conjunction with LQ control was tested for a range of weighing matrices Q_{LQR} and R_{LQR}. Table 1 contains an overview of the tested configurations. A unitary configuration was chosen as default, and the remaining selections are motivated by the experimental results described below.

An initial observation from the results is that the system now, in contrast to the open-loop implementation of section 1 successfully "follows" the optimal trajectory output by the open loop optimisation problem. This is natural and was expected, as the LQ controller introduces feedback to combat the effects of model errors and disturbances.

Furthermore, some interesting observations can be made that hold generally across trials. Firstly, although it approaches it, the travel never actually reaches zero. Similarly, the pitch levels out on a nonzero value even though the optimal trajectory would dictate otherwise. These two are no coincidence but are a direct consequence of inaccuracies in the model. Specifically, the systematic error discussed in section 1.6, means that keeping a constant travel requires a nonzero pitch. With this in mind, zero travel could be attained by simply having a larger stationary pitch. However, the LQ controller weighs the square of deviations of both pitch and travel. Hence, as

the pitch approaches a nonzero value needed to maintain zero travel, the control input becomes less and less incentivised to decrease the travel, because the square term in the pitch eventually dominates.

This effect persists as long as pitch is part of the weighted state vector. However, it can be minimised in favour of travel by weighing travel deviations much higher than those of pitch. The effect of this is seen by comparing the configuration of figure 7 with that in figure 8. Clearly, increasing the relative weight of the travel by 100 brings the travel much closer to its desired final value of zero, at the expense of a slightly higher stationary pitch value.

To demonstrate the general effect of weights on the input, or the pitch reference, relative to the state weights, figure 9 displays the effect of a somewhat extreme configuration where the pitch reference is weighted 1000 more than the unitary configuration in figure 7. The effect of this is that large pitch references are punished quite severely by the cost function. This has the important implication that the pitch necessary to keep a constant travel is no longer "worth it" for the LQ controller, such that the travel grows instead of approaching zero. An interesting theoretical extension of this is that as R_{LQR} approaches infinity, the state feedback gain matrix K from (10) approaches zero, meaning the system actually becomes the same open-loop system discussed in section 1!

To investigate the effect of weights on the travel rate r and pitch rate $\dot{p}$, an attempt was made to "optimise" the LQ weighing. The resulting configuration is shown in figure 10. A clear emphasis is placed on the travel, whose weight is set to a thousand. This augmentation alone produced aggressive pitch fluctuations, motivating the punishment of pitch rate by a weight of a hundred. Likewise, the weight on the travel rate was adjusted slightly up to prevent oscillations and overshoot. It is worth noting that we here "optimise" to fulfil the travel requirement of the optimal trajectory, which of course is at the expense of the other states and the input.

A final observation is that input constraint of $30°$ from the open loop optimisation problem in (8) is no longer satisfied. This is because the LQ controller synthesises the input from the state feedback and no longer simply uses the input from the optimisation problem. This is a possible disadvantage of LQ control as opposed to open loop optimisation which guaranteed that input constraints are satisfied. The clear advantages of LQ control, as discussed in the beginning of this section, can be considered to significantly outweigh this effect. A control method that achieves both these goals, MPC, was discussed in section 2.2.

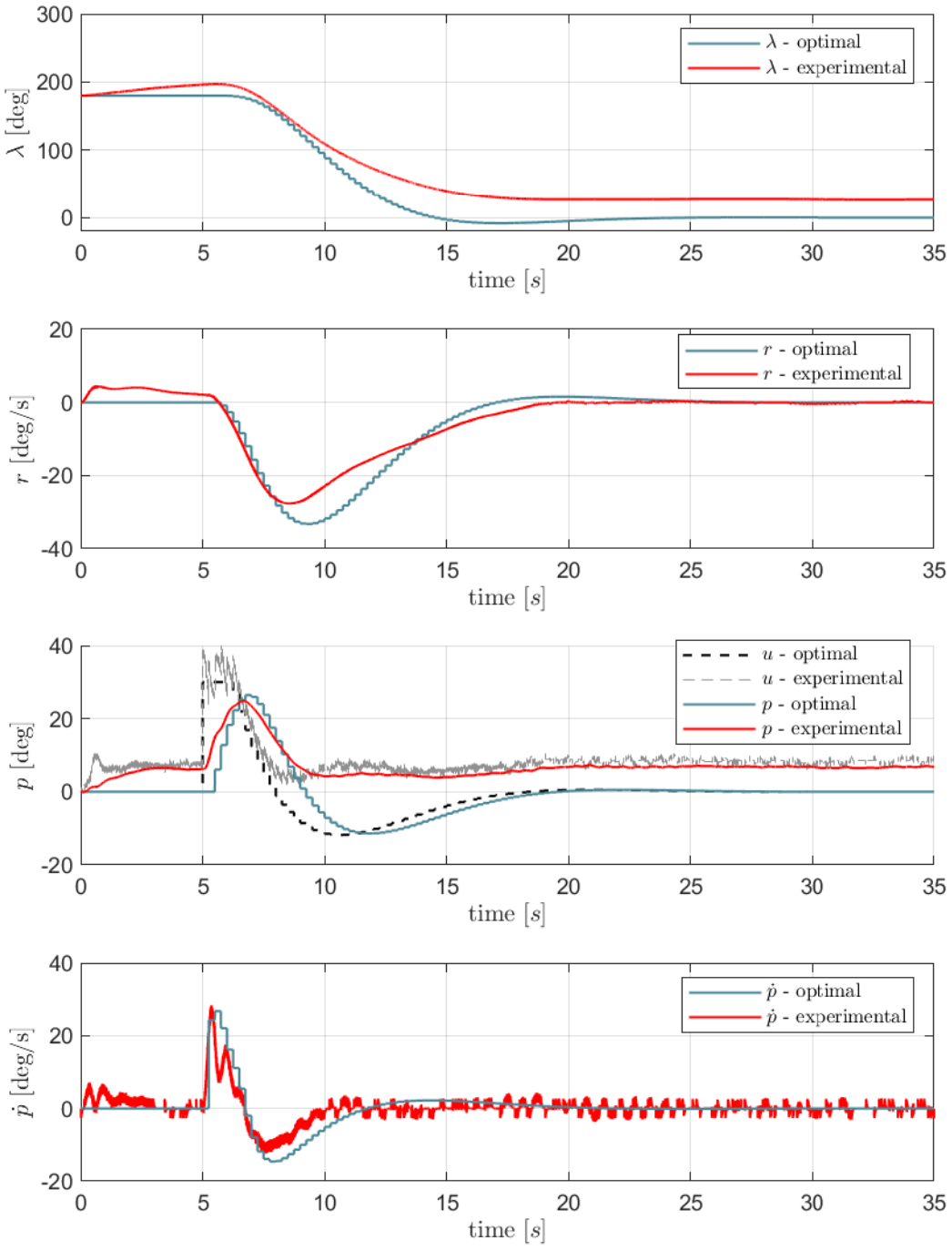


Figure 7: Results for diagonal elements of Q = [1 1 1 1] and R = 1.

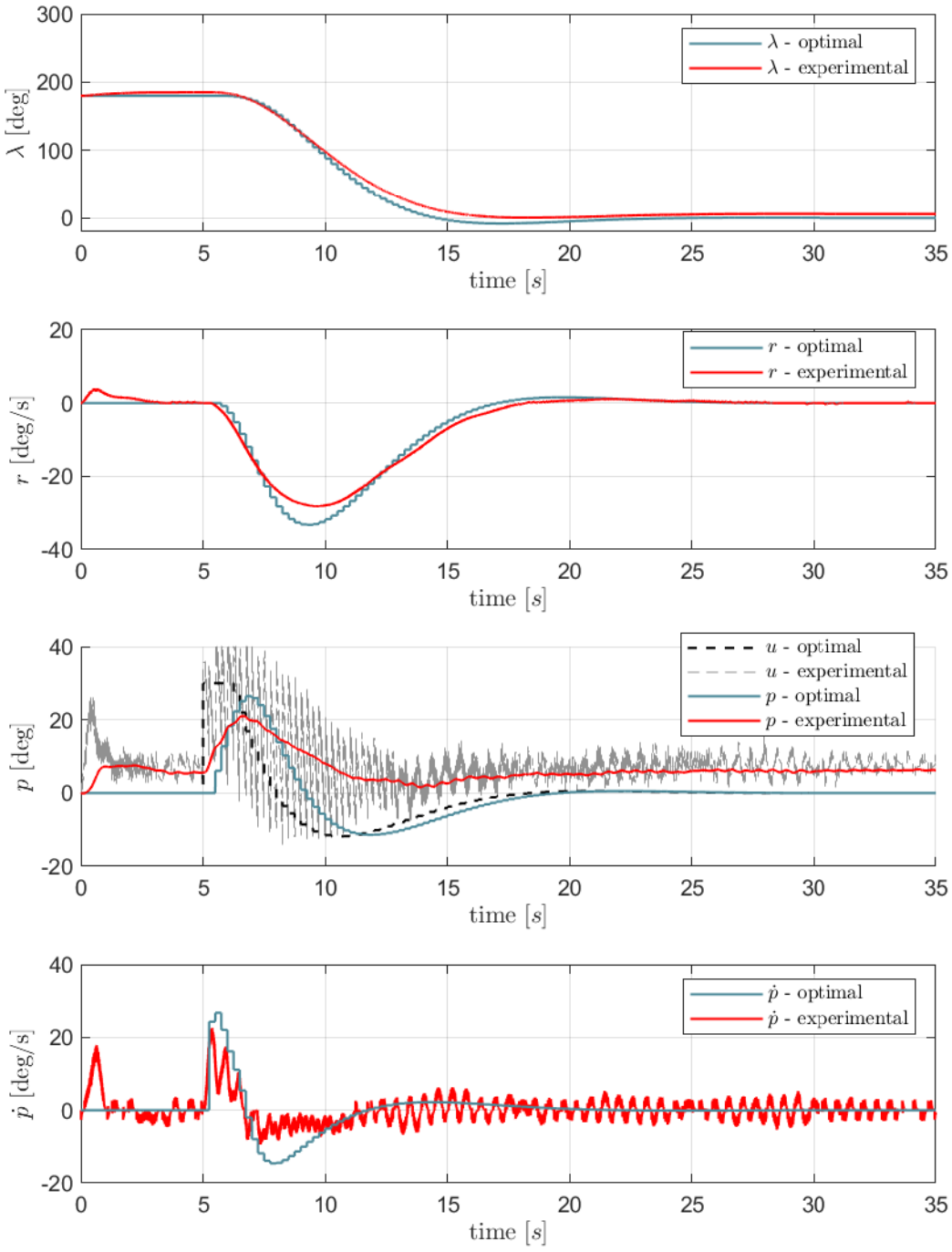
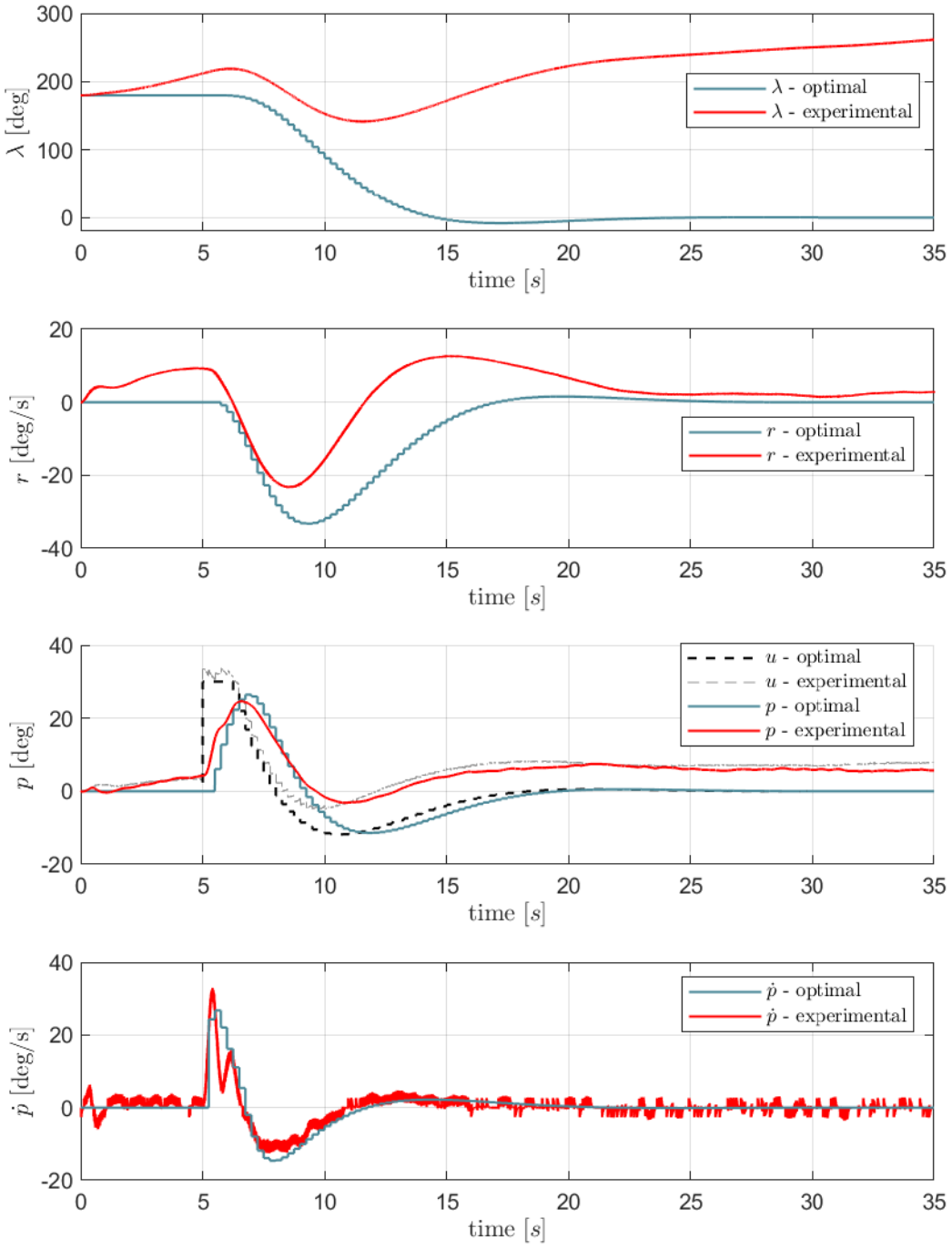


Figure 8: Results for diagonal elements of Q = [100 1 1 1] and R = 1.



gi

Figure 9: Results for diagonal elements of Q = [1 1 1 1] and R = 1000.

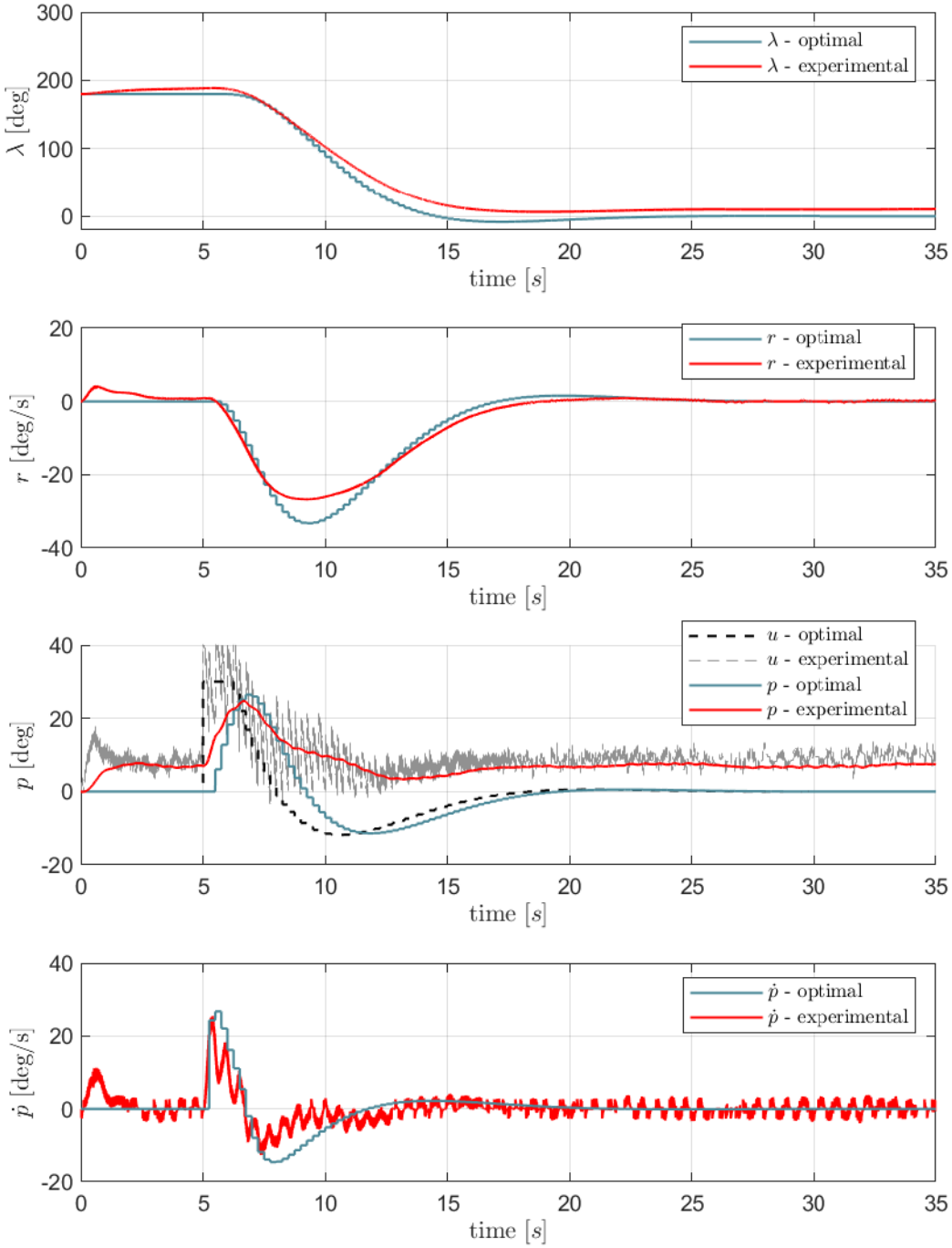


Figure 10: Results for diagonal elements of Q = [1000 10 1 100] and R = 1.

2.4 MATLAB and Simulink

Listing 2: The main Matlab code for this task is similar to the one from 1.7, but the gain matrix calculation has been added:

```
%% Discrete LQR
Q_lqr = diag([1 1 1 1]);
R_lqr = 1;
K_lqr = dlqr(A,B,Q_lqr,R_lqr);
```

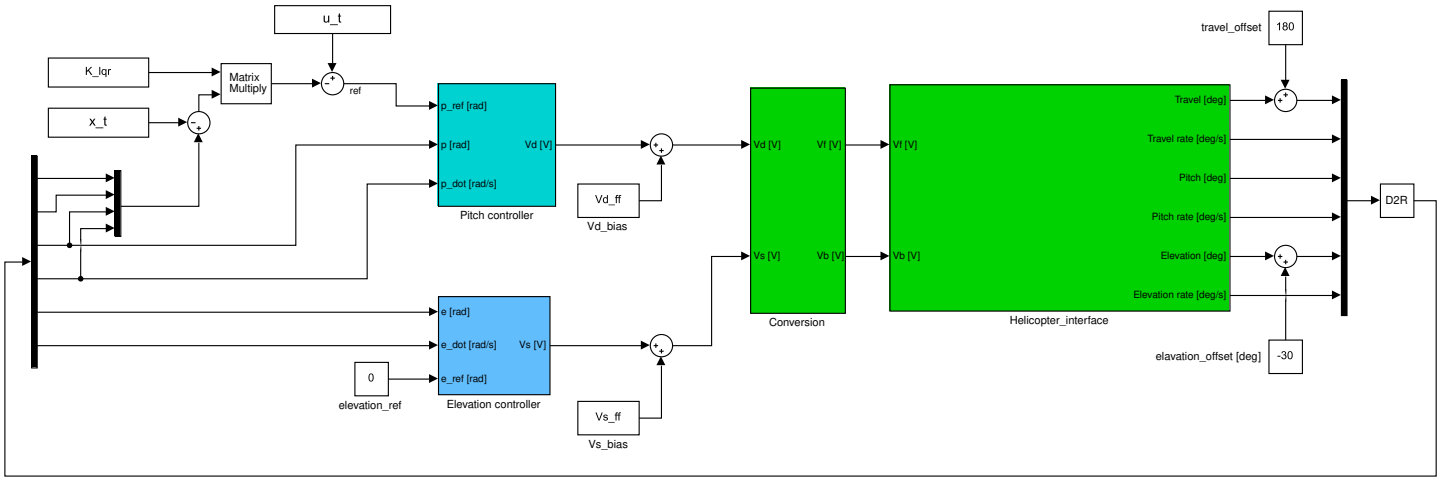


Figure 11: The simulink diagram for task 3. In the top left the LQ controller has been added.

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

In this section, we extend the control objective to steer the travel to its reference past an obstacle. This obstacle is modelled as a constraint on the elevation, in the sense that the helicopter along its trajectory must stay clear of the object. Because the elevation now is a central part of the problem, and a reference to elevation is needed as an explicit input, the model must be augmented.

3.1 The continuous model

The continuous dynamics are again given as equations in section 8 in the lab handout [1]:

$$\dot{\lambda} = r \tag{12a}$$
$$\dot{r} = -K_2 p \tag{12b}$$
$$\dot{p} = \dot{p} \tag{12c}$$
$$\ddot{p} = -K_1 K_{pp} p - K_1 K_{pd} \dot{p} + K_1 K_{pp} p_c \tag{12d}$$
$$\dot{e} = \dot{e} \tag{12e}$$
$$\ddot{e} = -K_3 K_{ep} e - K_3 K_{ed} \dot{e} + K_3 K_{ep} e_c \tag{12f}$$

They now include the dynamics for elevation and elevation rate, described by a PID controller for elevation. This controller was already implemented for convenience. With $x = [\lambda \ r \ p \ \dot{p} \ e \ \dot{e}]^\top$ and $u = [p_c \ e_c]^\top$, this can be expressed as

$$\dot{x} = A_c x + B_c u \tag{13}$$

where

$$\mathrm{A}_c = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3 K_{ep} & -K_3 K_{ed} \end{bmatrix}, \quad B_c = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1 K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3 K_{ep} \end{bmatrix}.$$

3.2 The discrete model

The discrete model is obtained by the forward Euler method described in 1. We set $A = I + \Delta t A_c$ and $B = \Delta t B_c$, which results in

$$x_{k+1} = Ax_k + Bu_k \tag{14}$$

where

$$A = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & -K_2\Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & -K_1K_{pp}\Delta t & 1 - K_1K_{pd}\Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & -K_3K_{ep}\Delta t & 1 - K_3K_{ed}\Delta t \end{bmatrix},$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp}\Delta t & 0 \\ 0 & 0 \\ 0 & K_3K_{ep}\Delta t \end{bmatrix}.$$

3.3 The optimisation problem

The control objective and state vector in this part of the project is a different one, which naturally calls for a redefinition of the optimisation problem. There are two modifications to the original problem from subsection 1.3. The first one is adding an extra term to the objective function, weighing the square of the new input, e_{ci}^2, such that it becomes:

$$\Phi = \sum_{i=0}^{N-1} (\lambda_{i+1} - \lambda_f)^2 + q_1 p_{ci}^2 + q_2 e_{ci}^2, \quad q_1, q_2 \geq 0. \tag{15}$$

This augmentation is natural and serves the same purpose as the weighing of p_{ci} described in section 1.3.

The second modification of the problem, is adding a nonlinear constraint on elevation to model the obstacle. Since the obstacle is spatially fixed, it is natural for this constraint to be a function of travel. It was given as

$$e_k \geq \alpha \exp(-\beta(\lambda_k - \lambda_t)^2) \quad \forall k \in 1, ..., N \tag{16}$$

where $\alpha = 0.2$, $\beta = 20$ and $\lambda_t = 2\pi/3 = 120°$. This constraint is illustrated by the dotted line in Figure 13, and evidently models a tall obstacle located at 120° travel. Due to the non-linearity of this constraint, the optimisation problem is no longer a quadratic program, and a nonlinear SQP solver was used to solve it. Refer to subsection 3.7 for implementational details.

3.4 Experimental results

The helicopter was controlled towards the trajectories output by the modified optimisation via an LQ controller with weighing matrices

$$Q = diag([100\ \ 1\ \ 1\ \ 10\ \ 1000\ \ 1])$$
$$R = diag([1\ \ 1]).$$

This was fixed throughout the trials, and focus was shifted to studying the impact of the input weights q_1 and q_2, as these are the only variables in the new optimisation problem. Table 2 summarises the tested values and their corresponding figures.

Table 2: Tested configurations of q_1 and q_2 and their corresponding figures.

$\mathbf{q_1}$	$\mathbf{q_2}$	**Figures**
1	1	Figure 12, Figure 13
10	1	Figure 14, Figure 15
100	100	Figure 17
0	500000	Figure 16

The optimal and experimental trajectories of the helicopter for $q_1 = 1, q_2 = 1$ are shown as functions of time in Figure 12. Looking at these trajectories, the travel successfully approaches zero, and the elevation is elevated slightly during the trajectory to overcome the obstacle. Because of the spacial nature of the constraint, this is much better analysed by plotting the elevation against travel, along with the constraint itself, which was done in Figure 13. It is evident that the optimal trajectory completely clears the obstacle at all *discrete* instances. Although it may seem like the constraint is violated around its peak, this is not in fact the case because the constraint, just as the optimisation problem, is discretely defined. It is also clear that the optimal trajectory is one that *just barely* clears the constraint at its peak. This is natural because higher elevations produce larger elevation references (inputs) without contributing to the travel goal. After overcoming the obstacle, the elevation reference quickly plummets to zero. This effect is only gradual for elevation because of the dynamics of the elevation in the model, and because the elevation itself is not weighted in the objective function.

Looking at the experimental trajectory, it is clear that it definitely violates the constraint in the vicinity of its peak. More generally, it seems to lie uniformly below the optimal elevation. This can be attributed to the LQ controller, which produces an input that tries to balance the requirements of all the states, and has no constraints itself. It could also result from in-

accuracies in the model.

In order to produce an optimal trajectory that better clears the constraint in the *continuous* sense, the weight q_1 was increased to 10. This increases the importance of minimising p_c, and decreases the travel speed required by the trajectory. This resulted in the trajectory shown in Figure 15, where both the optimal and experimental trajectories to a better job of clearing the obstacle. With this weighing, the pitch and travel trajectories still follow the optimal trajectories relatively closely, as seen in Figure 14.

To further test the impact of the discrete nature of the optimisation problem on the *continuous time* violation of constraints, the weighing parameters were set to $q_1 = 0$ and $q_2 = 500000$. The result is shown in Figure 16. By punishing e_c a lot and virtually leaving p_c unpunished, the optimisation algorithm chose to accelerate the helicopter such that it could take a larger discrete step when arriving at the constraint. This demonstrates that the right choice of q_1 and q_2 is crucial if a continuous constraint is to be followed. A solution to the issue could also be to decrease the time steps, but this has undesired effect on the size of the optimisation problem and hence performance. A more practical solution is to design the discrete constraint in a way that leaves some error margin to the actual obstacle. This also takes into account the error obtained from the experimental trajectory to the optimal.

Figure 16 also highlights a critical problem with the optimisation problem, namely that the trajectory no longer reaches the travel λ_f. If no constraint on the final state of λ is given (like $\lambda_N = \lambda_f$), too large weights will result in a optimal trajectory that simply remains on the initial side of the obstacle. An example of this is shown in Figure 17. Here, the needed elevation to overcome the obstacle is simply not worth the gain of approaching $\lambda = 0$. It is natural to consider this undesirable, and a solution could be giving the desired final state of the travel as a constraint to the optimisation problem.

As the previous experimental results have shown, no optimal elevation trajectory tightly followed the entire constraint. To investigate whether such a trajectory is even feasible for the helicopter, the constraint was imposed as an equality constraint. The resulting optimal and experimental trajectories are shown in Figure 18. It is clear that the optimisation problem can be solved to follow the trajectory tightly. The experimental trajectory, however, demonstrates that such a tight trajectory is infeasible for the LQ controller with the weights as selected. Furthermore, significant motor inputs are required to perform the acceleration and deceleration of elevation. It is therefore reasonable that such a tight trajectory is not optimal for any weights q_1 and q_2.

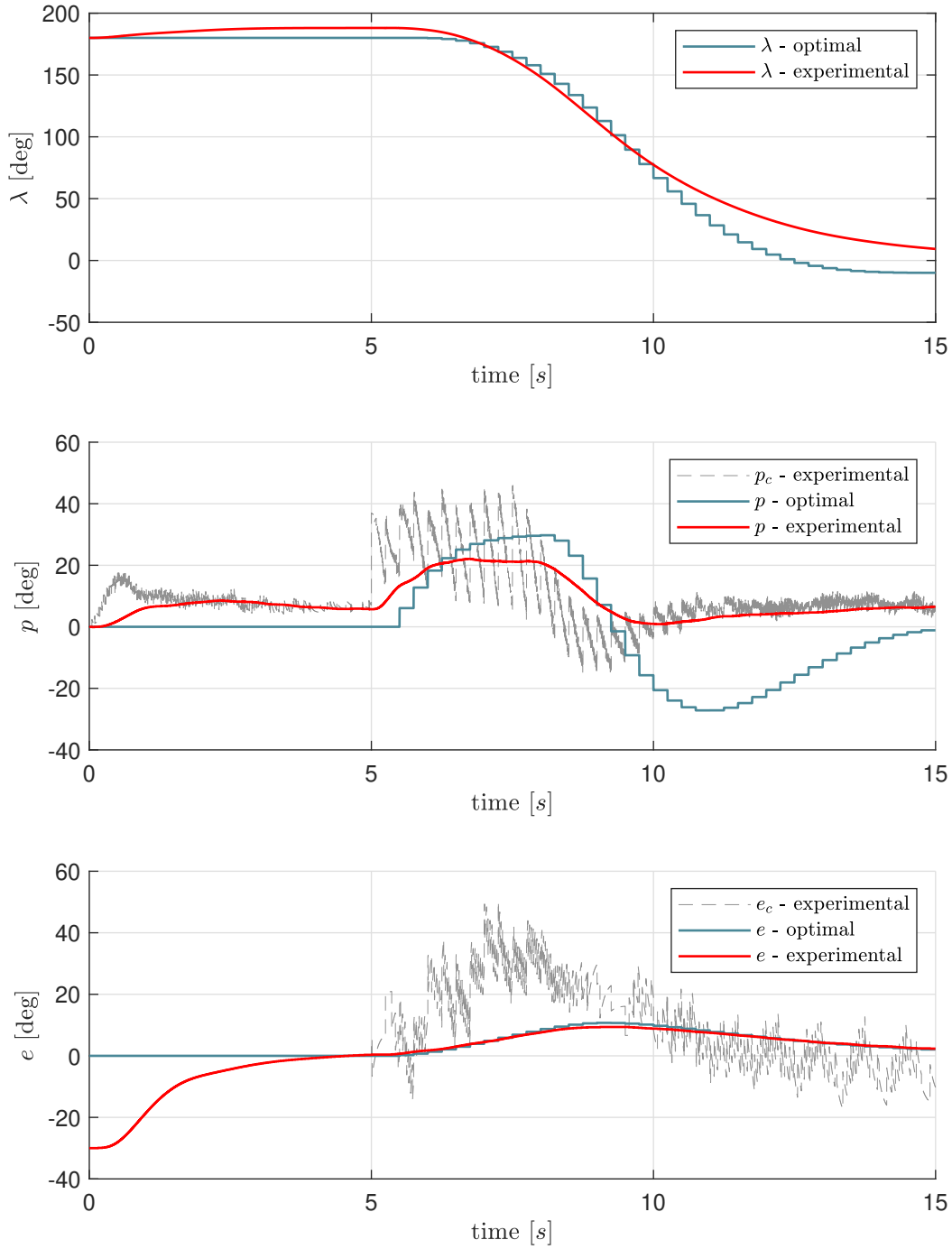


Figure 12: Optimal and experimental trajectories for $q_1 = 1$ and $q_2 = 1$.

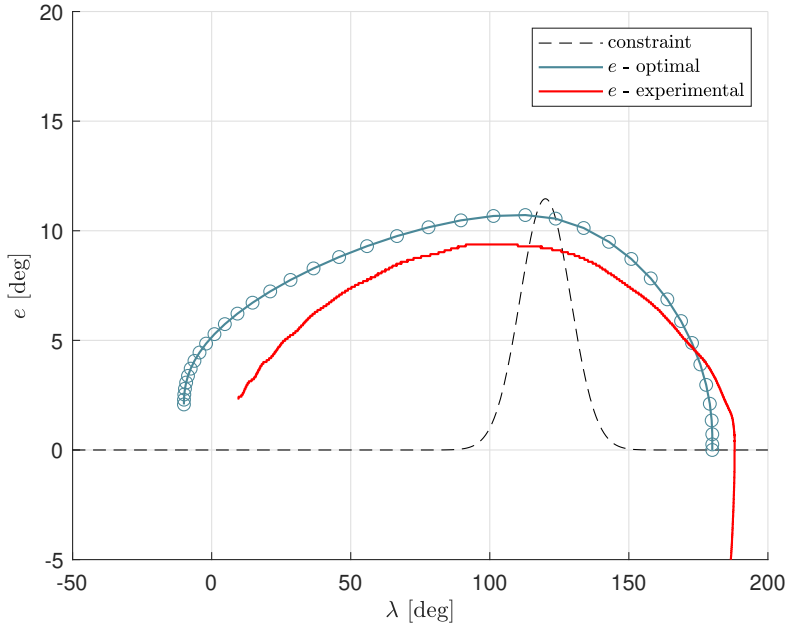


Figure 13: Elevation plottet against lambda for $q_1 = 1$ and $q_2 = 1$.

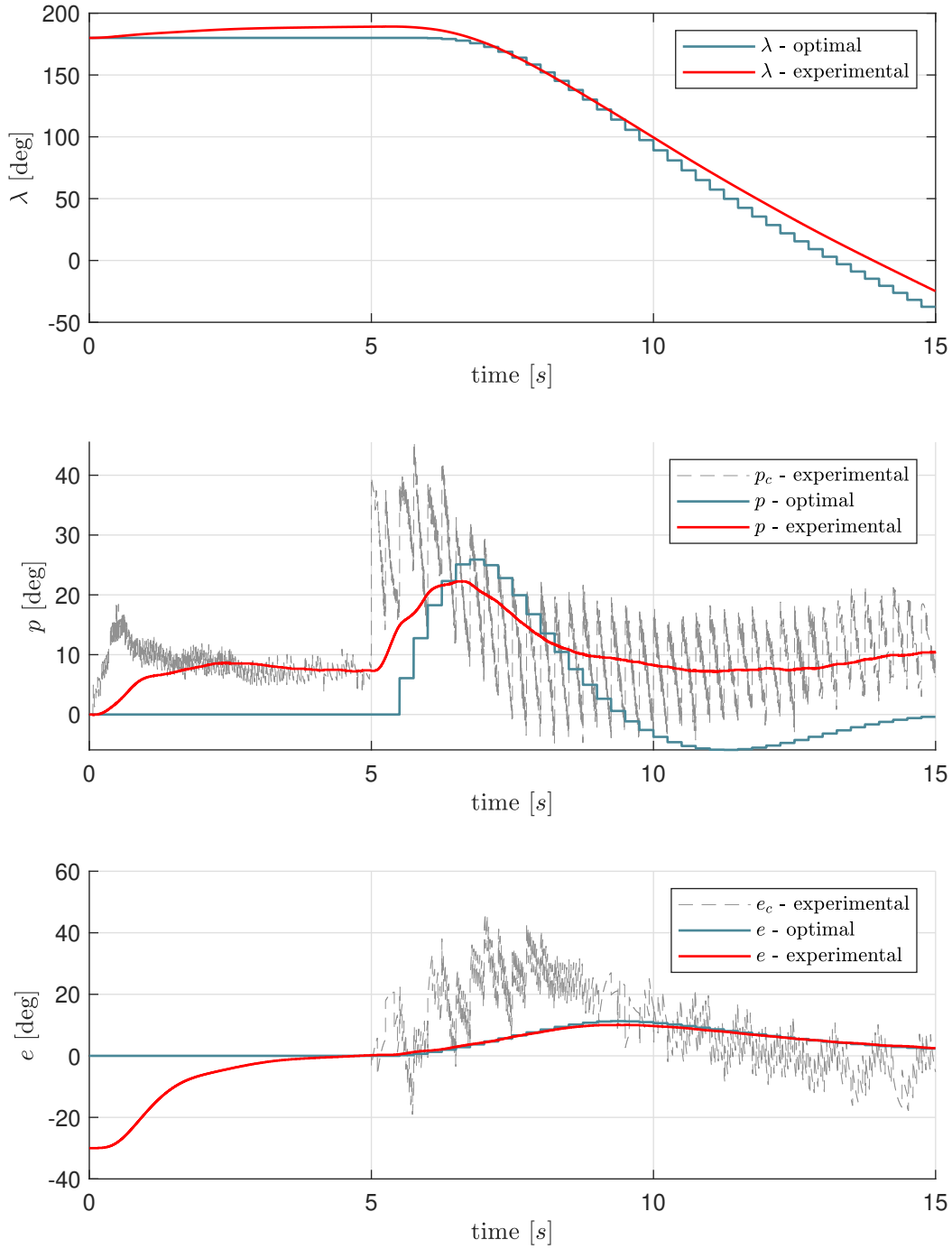


Figure 14: Optimal and experimental trajectories for $q_1 = 10$ and $q_2 = 1$.

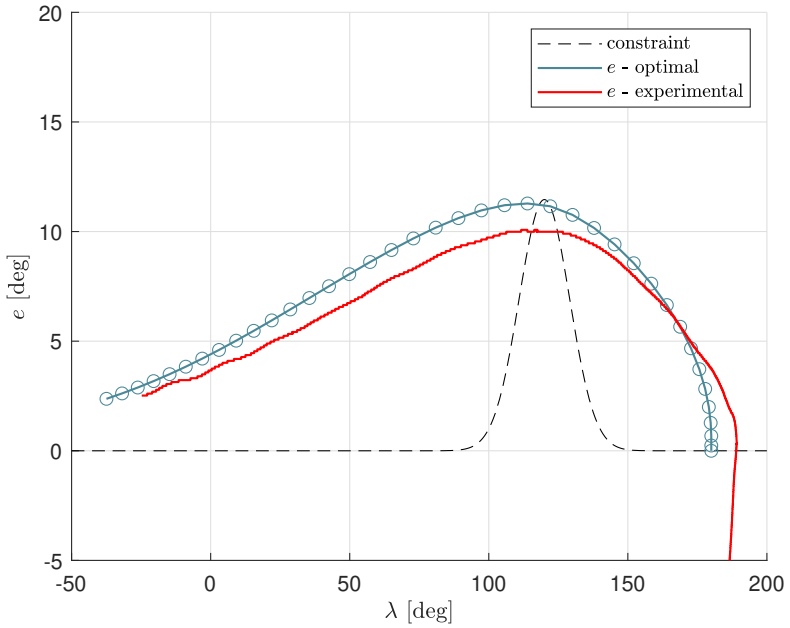


Figure 15: Elevation plottet against lambda for $q_1 = 10$ and $q_2 = 1$.

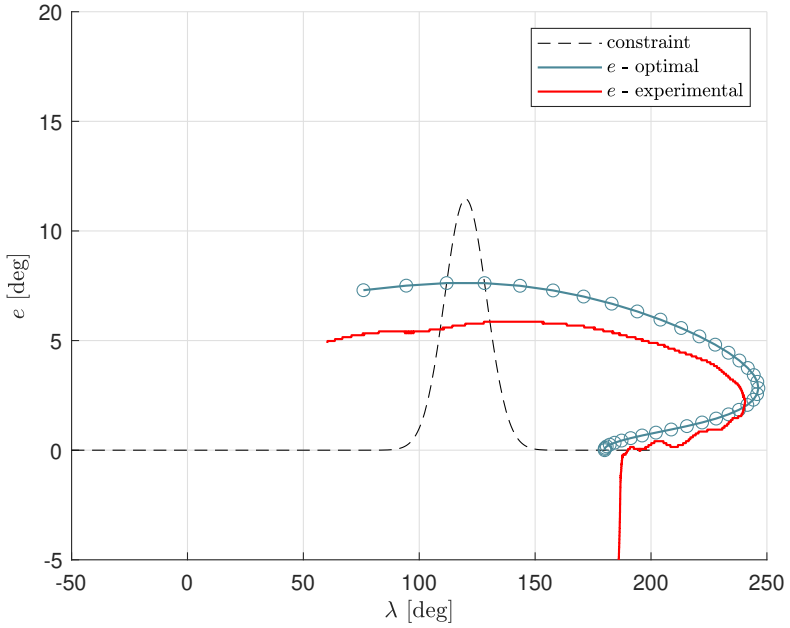


Figure 16: Elevation plottet against lambda for $q_1 = 0$ and $q_2 = 500000$.

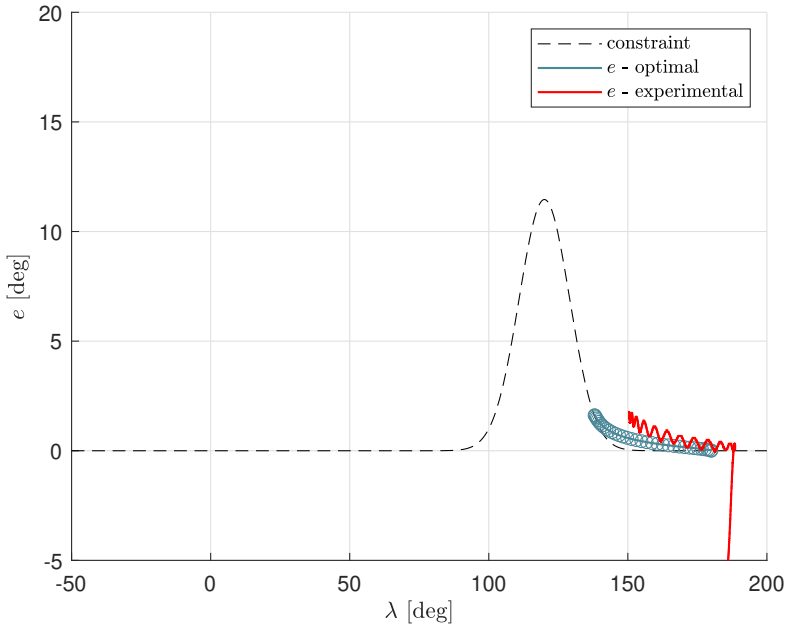


Figure 17: Elevation plottet against lambda for $q_1 = 100$ and $q_2 = 100$.

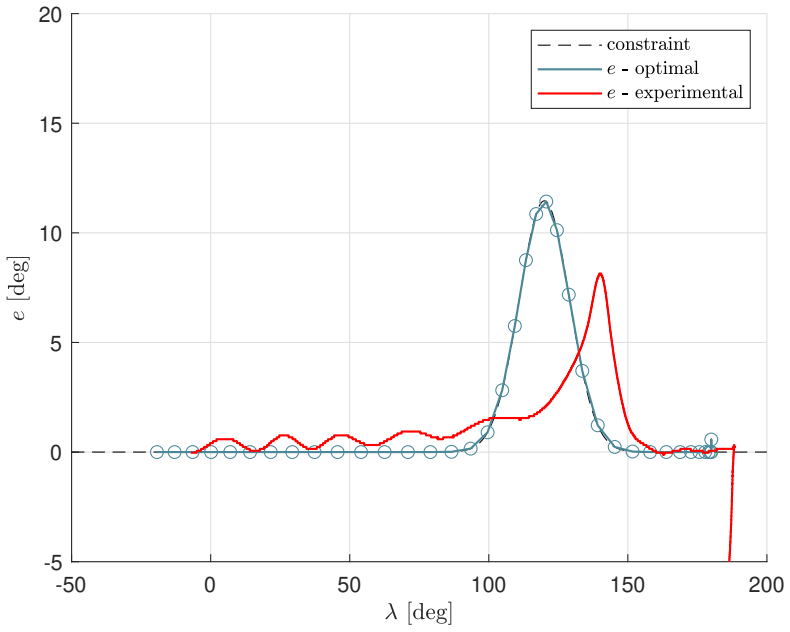


Figure 18: Elevation plottet against lambda for $q_1 = 1$ and $q_2 = 1$ with the nonlinear constraint set as an equality constraint.

3.5 Decoupled model

The discrete model 14 would, due to its block diagonal structure suggest that the four first states - pitch, pitch rate, travel and travel rate - are fully decoupled from the last two - elevation and elevation rate. Indeed, the continuous model equations 12 reflect this. These equations are however a linearisation of the "actual" nonlinear dynamics, derived in the course TTK4115 Linear Systems Theory:

$$J_p\ddot{p} = L_1 V_d \tag{17a}$$
$$J_e\ddot{e} = L_2 \cos(e) + L_3 V_s \cos(p) \tag{17b}$$
$$J_\lambda\ddot{\lambda} = L_4 V_s \cos(e) \sin(p) \tag{17c}$$

These nonlinear dynamics are, although not being perfect, a much better representation of the helicopter because they include the effect of the pitch angle on the vertical component of the voltage V_s for elevation, as well as the effect of the elevation angle on the horizontal component of V_s for the travel. This should effectively mean that the linearised model becomes a worse representation of elevation for increasing pitch angles, and a worse representation of travel for increasing elevation angles. This naturally leads the optimal trajectory to be incorrect in elevation and travel when the pitch is nonzero.

Interestingly enough, we observe that elevation control is quite good even for large pitch angles. The reason for this is that our LQR controller weights elevation much higher than pitch, and thus compensates for the model inaccuracy by giving a greater elevation input at the expense of pitch.

As for possible solutions, one solution could be to use a nonlinear controller using the complete nonlinear model. We would in this case of course have to change the entire optimisation problem, and could no longer readily use LQ control. An alternative solution could be to linearise the dynamics about multiple points, and partition the state space into regions described by different linear models. These models would however have to be switched between in run time, complicating the problem significantly.

3.6 Optional exercise

Since no motor time constant is modelled, constraints on $\dot{p}$ and $\dot{e}$ could be a beneficial addition to the optimisation problem, to prevent rapid changes in the input which the helicopter cannot follow. By studying the experimental

results from previous sections, these additional constraints were set to:

$$\begin{aligned} -5^\circ/s \leq \dot{p} \leq 5^\circ/s \\ -5^\circ/s \leq \dot{e} \leq 5^\circ/s \end{aligned} \tag{18}$$

Figure 20 displays the trajectories of λ, p and e. In comparison to Figure 12, both the optimal elevation and optimal pitch response are less aggressive in order to fulfil the constraints. Similarly, the experimental elevation and travel follow a smoother trajectory. Looking at the travel specifically, we observe a significant overshoot at the end of the trajectory. This is a consequence of the introduced damping, and could be considered a slight disadvantage of the added constraints.

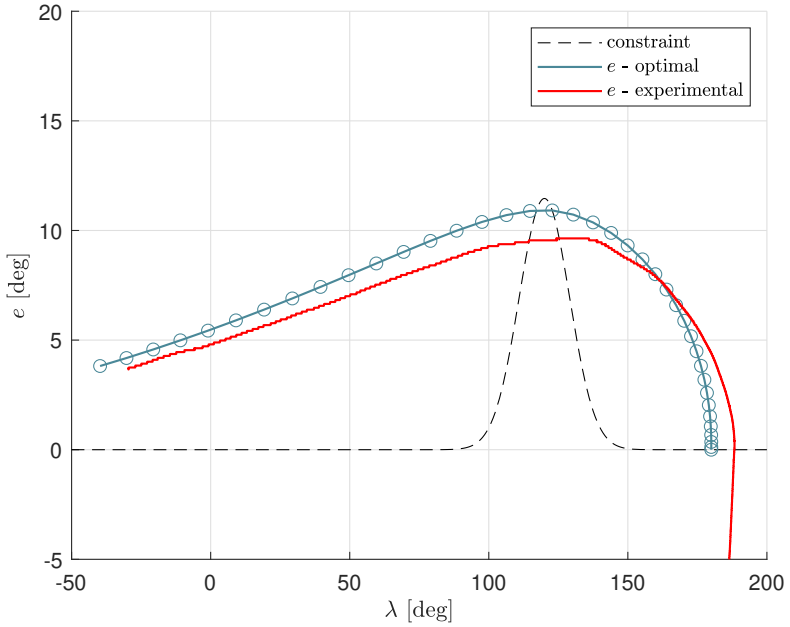


Figure 19: Elevation plottet against lambda for $q_1 = 1$ and $q_2 = 1$ and with constraints on $\dot{e}, \dot{p} \in [-5^\circ/s, 5^\circ/s]$.

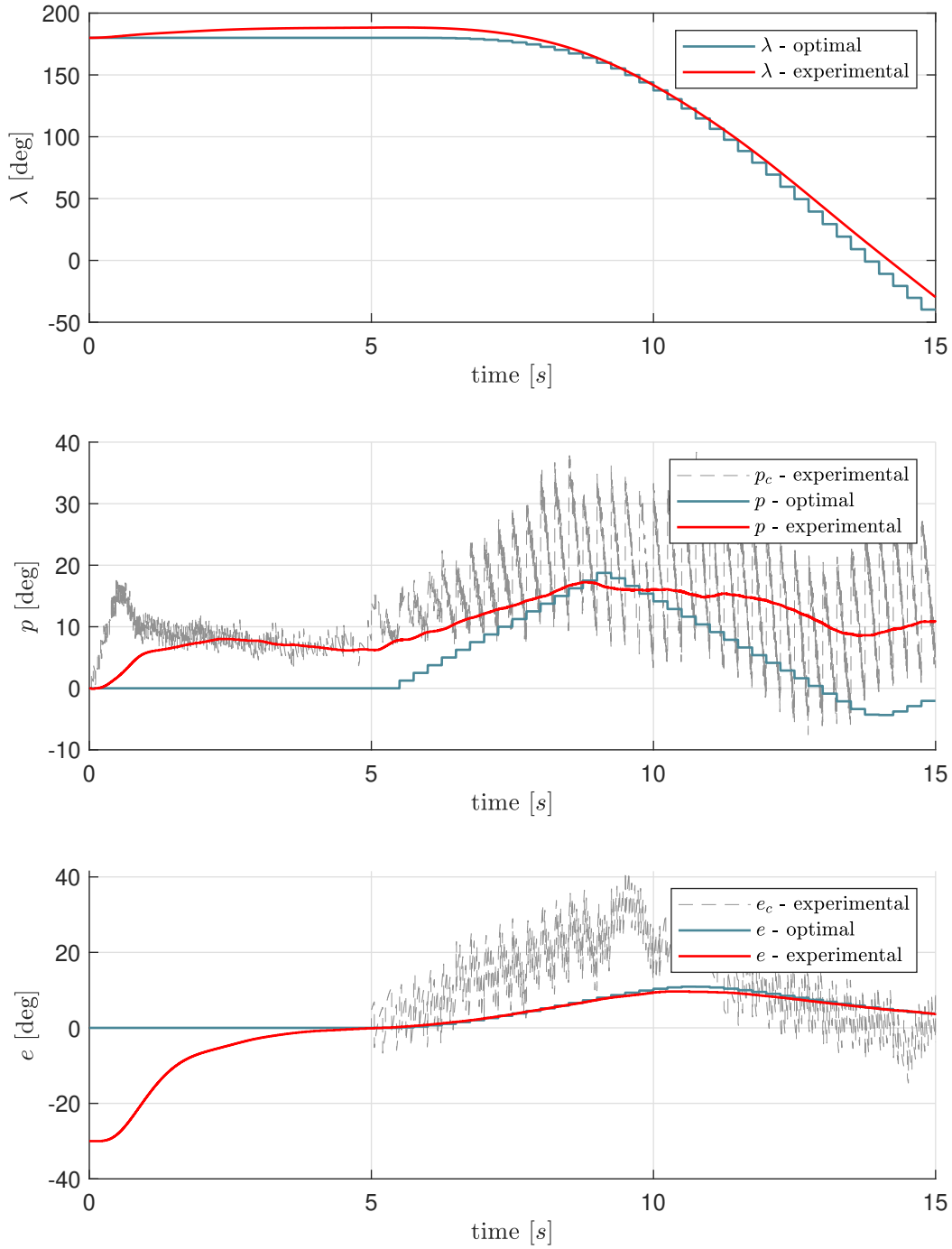


Figure 20: Optimal and experimental trajectories for $q_1 = 1$ and $q_2 = 1$ and with constraints on $\dot{e}, \dot{p} \in [-5°/s, 5°/s]$.

3.7 MATLAB and Simulink

Listing 3: The main matlab code for task 4.

```
%% Initialization and model definition
init05;

% Discrete time system model. x = [lambda r p p_dot]'
delta_t = 0.25; % sampling time
A = [1 delta_t 0 0 0;
     0 1 -delta_t*K_2 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 -K_3*K_ep*delta_t 1-K_3*K_ed*delta_t];
B = [0 0; 0 0; 0 0; K_1*K_pp*delta_t 0; 0 0; 0 K_3*K_ep*delta_t];

% 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
u0 = [ 0; 0];
% 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
z0(1:mx,:) = x0;

% Bounds
alpha = 0.2;
beta = 20;
lambda_t = 2*pi/3;

ul          = [-30*pi/180; -inf];           % Lower bound on control
uu          = [30*pi/180; +inf];            % Upper bound on control

xl      = -Inf*ones(mx,1);                  % Lower bound on states (no bound)
xu      = Inf*ones(mx,1);                   % Upper bound on states (no bound)

xl(3)   = -30*pi/180;                       % Lower bound on state x3
xu(3)   = 30*pi/180;                        % Upper bound on state x3


% xl(4)   = -5*pi/180;                      % Optinal lower bound on state x4
% xu(4)   = 5*pi/180;                       % Optinal upper bound on state x4

% xl(6)   = -5*pi/180;                      % Optinal lower bound on state x6
% xu(6)   = 5*pi/180;                       % Optinal upper bound on state x6

% Generate constraints on measurements and inputs
[vlb,vub]       = gen_constraints(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
```

```
59
60  vlb(N*mx-5) = 0;
61  vub(N*mx-5) = 0;
62
63  nonlcon = @(x,N,mx)elev_con;
64
65  % Generate the matrix Q and the vector c (objecitve fun0ction weights in the QP
        problem)
66  Q = zeros(mx,mx);
67  Q(1,1) = 1;                                 % Weight on state x1
68  Q(2,2) = 0;                                 % Weight on state x2
69  Q(3,3) = 0;                                 % Weight on state x3
70  Q(4,4) = 0;                                 % Weight on state x4
71  Q(5,5) = 0;                                 % Weight on state x5
72  Q(6,6) = 0;                                 % Weight on state x6
73
74  q1 = 1;                                     % Weight on input
75  q2 = 1;
76  R = diag([q1 q2]);
77  G = gen_q(Q,R,N,M);
78
79  objective = @(x) x'*(G*x);
80
81  %% Generate system matrixes for linear model
82  Aeq = gen_aeq(A,B,N,mx,mu);
83  beq = zeros(N*length(x0),1);
84  beq(1:length(x0)) = A*x0;
85
86
87  %% Solve QP problem with nonlinear constraint
88  options = optimoptions('fmincon','Display','iter','Algorithm','sqp','
        MaxFunctionEvaluations',3.2e+9);
89  tic
90  [z,fval] = fmincon(objective,z0,[],[],Aeq,beq,vlb,vub, @(x)elev_con(x,N,mx,
        alpha, beta, lambda_t),options);
91  t1=toc;
92
93  % Calculate objective value
94  phi1 = 0.0;
95  PhiOut = zeros(N*mx+M*mu,1);
96  for i=1:N*mx+M*mu
97    phi1=phi1+Q(i,i)*z(i)*z(i);
98    PhiOut(i) = phi1;
99  end
100
101 %% Extract control inputs and states
102 u1  = [z(N*mx+1:mu:N*mx+M*mu);z(N*mx+M*mu-1)];
103 u2  = [z(N*mx+2:mu:N*mx+M*mu);z(N*mx+M*mu)];
104
105 x1 = [x0(1);z(1:mx:N*mx)];                  % State x1 from solution
106 x2 = [x0(2);z(2:mx:N*mx)];                  % State x2 from solution
107 x3 = [x0(3);z(3:mx:N*mx)];                  % State x3 from solution
108 x4 = [x0(4);z(4:mx:N*mx)];                  % State x4 from solution
109 x5 = [x0(5);z(5:mx:N*mx)];
110 x6 = [x0(6);z(6:mx:N*mx)];
111
112 num_variables = 5/delta_t;
113 zero_padding = zeros(num_variables,1);
114 unit_padding  = ones(num_variables,1);
115
116 u1   = [zero_padding; u1; u1(end)*ones(num_variables,1)];
117 u2   = [zero_padding; u2; u2(end)*ones(num_variables,1)];
```

```
x1  = [pi*unit_padding; x1; x1(end)*ones(num_variables,1)];
x2  = [zero_padding; x2; x2(end)*ones(num_variables,1)];
x3  = [zero_padding; x3; x3(end)*ones(num_variables,1)];
x4  = [zero_padding; x4; x4(end)*ones(num_variables,1)];
x5  = [zero_padding; x5; x5(end)*ones(num_variables,1)];
x6  = [zero_padding; x6; x6(end)*ones(num_variables,1)];

x_with_padding = [x1; x2; x3; x4; x5; x6];
%% Time series input
time_steps = [0:delta_t:delta_t*(length(u1)-1)]';
u1_t = timeseries(u1,time_steps);
u2_t = timeseries(u2,time_steps);
x1_t = timeseries(x1, time_steps);
x2_t = timeseries(x2, time_steps);
x3_t = timeseries(x3, time_steps);
x4_t = timeseries(x4, time_steps);
x5_t = timeseries(x5, time_steps);
x6_t = timeseries(x6, time_steps);

x = [x1 x2 x3 x4 x5 x6];
x_t.time = time_steps;
x_t.signals.values = x;
x_t.signals.dimensions = mx;

%% Solve Discrete LQR
Q_lqr = diag([100 1 1 10 1000 1]);
R_lqr = diag([1 1]);
K_lqr = dlqr(A,B,Q_lqr,R_lqr);
```

Listing 4: The implementation of the nonlinear constraint.

```
function [c,ceq] = elev_con(x, N,mx, alpha, beta, lambda_t)
c = zeros(N,1);
for i = 0:N-1
    c(i+1) = alpha*(exp(-beta*(x(mx*i+1)-lambda_t)^2))-x(mx*i+5);
end
ceq= [];
end
```

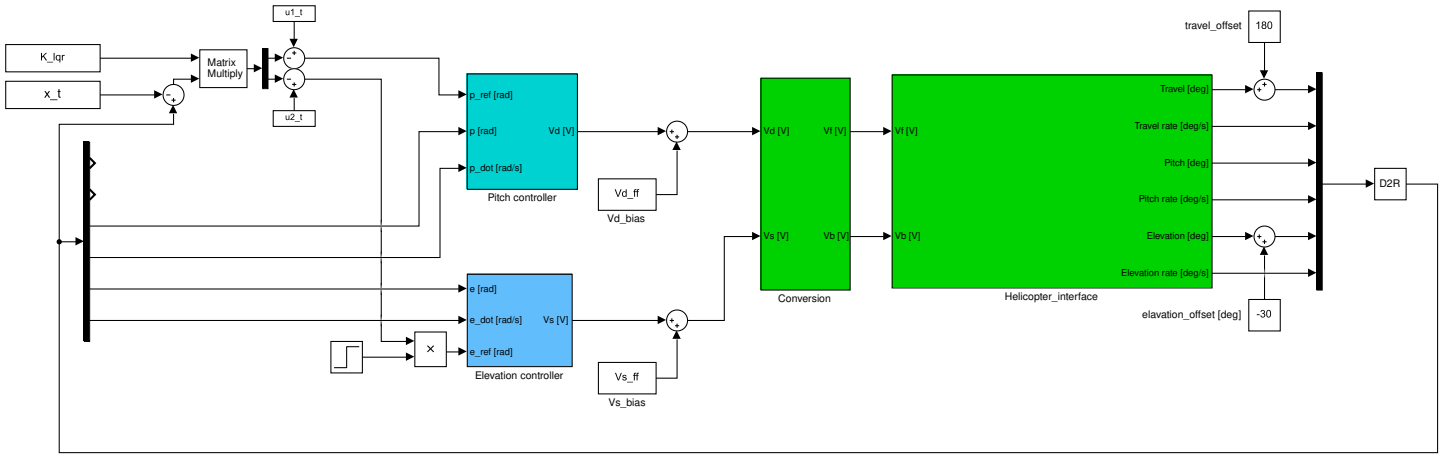


Figure 21: Simulink diagram for task 4. The step input on the elevation is implemented to let the helicopter reach 0 elevation before the lqr controller begins.

References

[1] Ttk4135 optimization and control, helicopter lab. TTK4135 Blackboard. Accessed: Spring 2022.

[2] Lars Imsland. Ttk4135 – lecture 8, open-loop dynamic optimization. TTK4135 - Blackboard. Accessed: Spring 2022.