Lab Report

Student 507718
Student 506878

April 07, 2021

Department of Engineering Cybernetics

Contents

1 10.2 - Optimal Control of Pitch/Travel without feedback . . . 1
- 1.1 The continuous model . . . 1
- 1.2 The discretized model . . . 3
- 1.3 The open loop optimization problem . . . 3
- 1.4 The weights of the optimization problem . . . 5
- 1.5 The objective function . . . 5
- 1.6 Experimental results . . . 5
- 1.7 MATLAB and Simulink . . . 8

2 10.3 - Optimal Control of Pitch/Travel with Feedback (LQ) . . . 12
- 2.1 LQ controller . . . 12
- 2.2 Model Predictive Control . . . 12
- 2.3 Experimental results . . . 13
- 2.4 MATLAB and Simulink . . . 16

3 10.4 - Optimal Control of Pitch/Travel and Elevation with Feedback . . . 21
- 3.1 The continuous model . . . 21
- 3.2 The discretized model . . . 21
- 3.3 Experimental results . . . 22
- 3.4 Decoupled model . . . 22
- 3.5 Optional exercise . . . 25
- 3.6 MATLAB and Simulink . . . 26

References . . . 31

1 10.2 - Optimal Control of Pitch/Travel without feedback

1.1 The continuous model

The task is to calculate and implement an optimal input trajectory x^* and a corresponding optimal input sequence u^* in order to complete a travel rotation of 180°. There is no feedback of the measured states.

In order to easily manipulate the helicopter's behaviour, the continuous state space model is written as follows:

$$\dot{\boldsymbol{x}} = \boldsymbol{A_c}x + \boldsymbol{B_c}u \tag{1}$$

with state vector x and input u defined as

$$x = \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \end{bmatrix}, u = p_c \,. \tag{2}$$

The state vector consists of the travel angle, λ, speed of travel, r, pitch position, p, and pitch velocity, $\dot{p}$. As we disregard the elevation, we assume $e = 0$ and thereby have a mono variable input given by p_c. These are the states we are modelling. The model is given through the following equations from the labexercise manual [1].

$$\ddot{e} + K_3K_{ed}\dot{e} + K_3K_{ep}e = K_3K_{ep}e_c \tag{3a}$$

$$\ddot{p} + K_1K_{pd}\dot{p} + K_1K_{pp}p = K_1K_{pp}p_c \tag{3b}$$

$$\dot{\lambda} = r \tag{3c}$$

$$\dot{r} = -K_2p \tag{3d}$$

When putting these on form of eq. (1) and disregarding eq. (3a) $\mathbf{A_c}$ and $\mathbf{B_c}$ becomes

$$\boldsymbol{A}_c = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} \end{bmatrix}, \boldsymbol{B_c} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1K_{pp} \end{bmatrix} \tag{4}$$

where frictions are neglected and the model is linearized.

In the *Basic control layer* in fig. 1, we only need a PD-regulator, not a PID as we disregard the elevation. This also makes the input a scalar, consisting of p_c, the set point for the pitch angle. Our system describes the entirety of fig. 1 excluding the optimization layer.

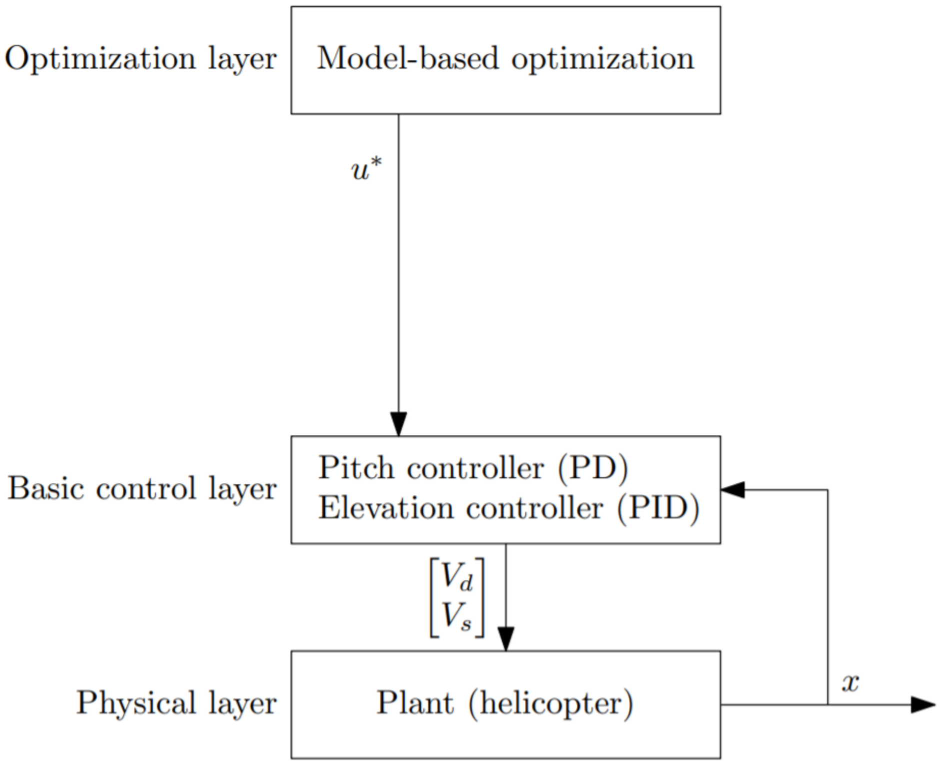


Figure 1: Illustration of the layers in the control hierarchy.

1.2 The discretized model

As the control inputs are to be given at time steps $k \in [0,...,N-1]$ we need to discretize our state space model. The forward Euler method is used to descretize with the following approximation

$$\frac{x_{k+1}-x_k}{\Delta t} = \boldsymbol{A_c} x_k + \boldsymbol{B_c} u_k. \tag{5}$$

Solving this equation for $\mathbf{x}_{\mathrm{k+1}}$ provides the relation

$$x_{k+1} = \boldsymbol{A_d} x_k + \boldsymbol{B_d} u_k \tag{6}$$

with the new discrete system matrices:

$$\boldsymbol{A_d} = (\Delta t \boldsymbol{A_c} + \boldsymbol{I}) \tag{7}$$

$$\boldsymbol{B_d} = \Delta t \boldsymbol{B_c} \tag{8}$$

Inserting the continuous system matrices from eq. (4) yields

$$\boldsymbol{A_d} = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & -\Delta t K_2 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & -\Delta t K_1 K_{pp} & 1-\Delta t K_1 K_{pd} \end{bmatrix} \boldsymbol{B_d} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \Delta t K_1 K_{pp} \end{bmatrix}. \tag{9}$$

1.3 The open loop optimization problem

We are now to calculate an optimal trajectory for the helicopter using the discretized model. The narrative is to move the helicopter from the initial value from $x_0 = \begin{bmatrix} \lambda_0 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$ to $x_f = \begin{bmatrix} \lambda_f & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$. This means finding the sequence of control input signals which will minimize the cost function of pivoting the helicopter 180°.

The QP problem we want to minimize:

$$\phi = \sum_{i=0}^{N-1} (\lambda_{i+1} - \lambda_f)^2 + q p_{ci}^2. \tag{10}$$

with $q \geq 0$. We also have the constraint

$$|p_k| \leq \frac{30\pi}{180}, k \in (1,...,N) \tag{11}$$

Since this is a quadratic cost function with linear constraints it can be reformulated to standard form

$$\min_{z \in \mathbb{R}^{5N}} \frac{1}{2} z^\mathsf{T} \boldsymbol{Q} z \text{ such that } \begin{cases} \boldsymbol{A}_{eq} z = b_{eq} \\ u_l \leq u_k \leq u_u \\ x_l \leq x_k \leq x_u \end{cases}, z = [x_1^T,...,x_N^T,u_0,...,u_{N-1}^T] \tag{12}$$

The vector z contains all predicted states x_k and their corresponding input signals u_{k-1} for all the time steps k in the event horizon of $N = 100$. The diagonal matrix $\boldsymbol{Q}$ is constructed such that the elements on the diagonal represents the states present in the objective function, eq. (10), as well as the weight of the control input limit.

$$\boldsymbol{Q} = \begin{bmatrix} Q_1 & & & & & \\ & \ddots & & & & \\ & & Q_1 & & & \\ & & & P_1 & & \\ & & & & \ddots & \\ & & & & & P_1 \end{bmatrix} \tag{13}$$

The variables are expressed through the 4x4 matrix $\boldsymbol{Q}_1$. P_1 represents the weight of the control input limitation. We can use the built in convex QP solver `quadprog` in Matlab to solve this QP problem with linear constraints.

As λ is the only state variable present in the cost function, the matrix $\boldsymbol{Q}_1$ is zero everywhere but the first element. As the standard form of the quadratic cost function contains a factor of $1/2$, $\boldsymbol{Q}_1$ results in

$$\boldsymbol{Q}_1 = \begin{bmatrix} 2 & & & \\ & 0 & & \\ & & 0 & \\ & & & 0 \end{bmatrix} \quad . \tag{14}$$

Section 1.7 shows the implementation of the code solving the optimization problem. The first lines generates the system matrices from eq. (6) as the equations for each time step can be written out as a set of linear equations:

$$-ax_0 + x_1 - bu_0 = 0 \tag{15a}$$
$$-ax_1 + x_2 - bu_2 = 0 \tag{15b}$$
$$\vdots \tag{15c}$$
$$-ax_{N-1} + N - bu_{N-1} = 0 \tag{15d}$$

using z from eq. (12), yields the system

$$\boldsymbol{A}_{eq}z - \boldsymbol{b}_{eq} = 0 \quad , \tag{16}$$

$$\boldsymbol{A}_{eq} = \left[\begin{array}{cccc|ccc} \boldsymbol{I} & & & & -\boldsymbol{B} & & \\ -\boldsymbol{A} & \boldsymbol{I} & & & & \ddots & \\ & \ddots & \ddots & & & & \ddots \\ & & -\boldsymbol{A} & \boldsymbol{I} & & & -\boldsymbol{B} \end{array}\right], \boldsymbol{b}_{eq} = \begin{bmatrix} \boldsymbol{A}x_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{17}$$

Equation (17) represents the matrices inserted into the quadprog function. The blank elements in $\boldsymbol{A}_{eq}$ represents zero-block matrices. The zero-padding is due to the zero reference when the helicopter flies in equilibrium. The full Matlab and Simulink implementation is presented in section 1.7

1.4 The weights of the optimization problem

Experimenting with different values of P_1 yields different penalization on the input signal and different behaviour of the helicopter. By limiting the control input signal we may prevent using rash and sudden changes to the pitch angle. The response for the state variables and input with values of P_1 being 0.1, 1 and 10 are shown in fig. 2a, fig. 2b and fig. 3.

With P_1 = 0.1 the helicopter moved rapidly and with sudden changes, this is because the input signal is penalized less. With P_1 = 10 the helicopter moves slower and with less precision. We found that P_1 = 1 gave the best performance, as shown in fig. 2b and fig. 4 because the helicopter diverged the least from the trajectory.

1.5 The objective function

Since λ_f = 0, equation eq. (10) becomes

$$\phi = \sum_{i=0}^{N-1} (\lambda_{i+1})^2 + qp_{ci}^2, q \geq 0 \tag{18}$$

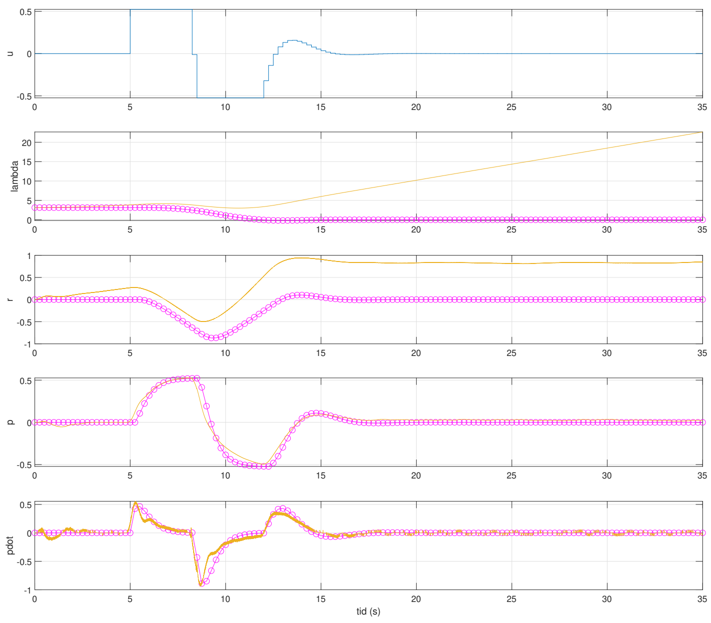
When steering the helicopter to $\lambda = \lambda_f$ with this objective function we get

$$\phi = \sum_{i=0}^{N-1} qp_{ci}^2 \tag{19}$$

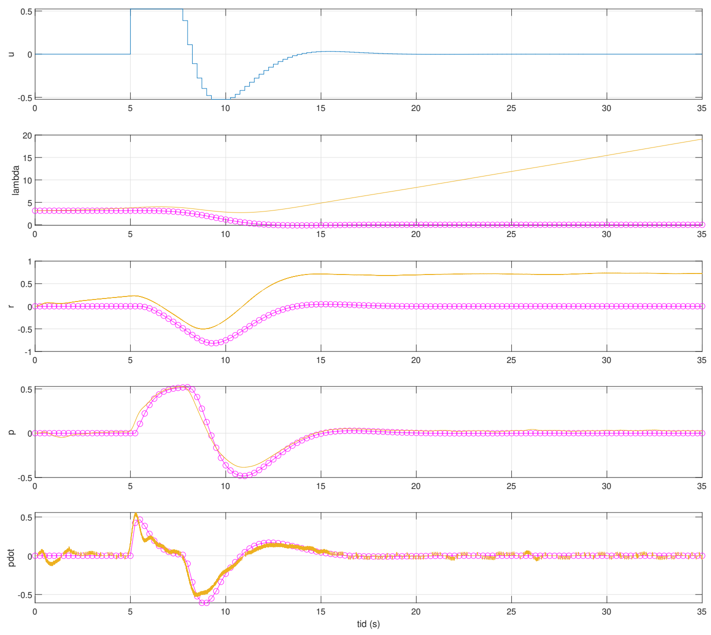
The input signal is eliminated. The system assumes the pitch angle is 0, but a minor measurement error or deviation will cause the helicopter to drift off.

1.6 Experimental results

The input trajectory seem reasonable based on the goal of the task, to turn the helicopter 180°. Figure 2b shows the trajectory with the optimal P. The code used to find this trajectory, and the Simulink we implemented are shown in section section 1.7. The helicopter does not end in the desired point λ_f, as we can see in



(a) P = 0.1



(b) P = 1

Figure 2: Different values for P

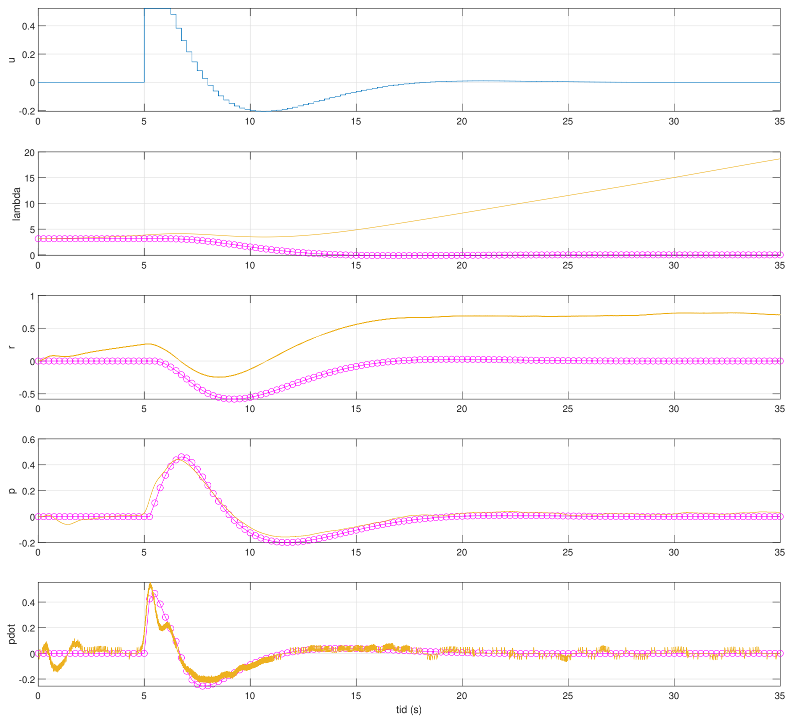


Figure 3: P = 10

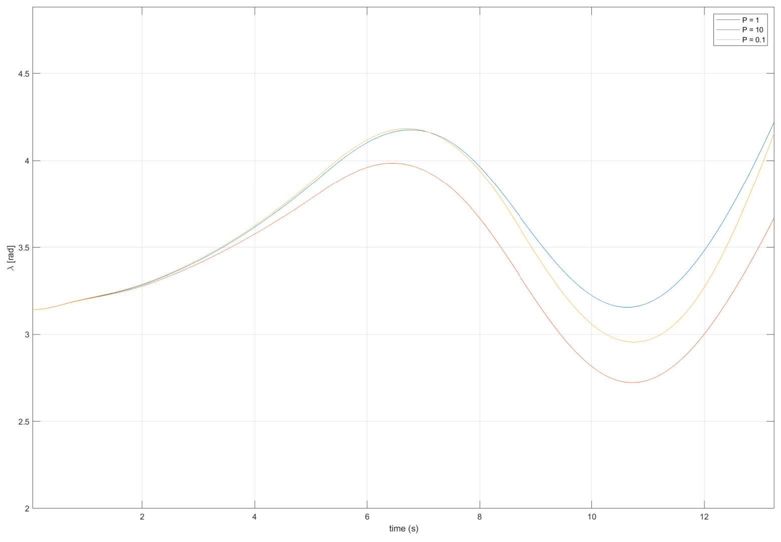


Figure 4: Travel angles, using different values for P.

fig. 2b. This is because the model does not take travel rate into consideration and therefore the chances of a full stop when the helicopter reaches $\lambda = \lambda_f$ are small. With an open-loop solution, the helicopter has no way of knowing if it follows the optimal trajectory and continues to drift off. The helicopter brakes by reversing the pitch and because of the lack of feedback, the helicopter does not know when it has reached x_f. The difference between the pitch trajectory and the pitch measurement are due to this. This results in the helicopter passing the target at $\lambda = \lambda_f$ and not noticing that is has reached its desired position. The conclusion is that our model is not good enough to achieve the desired state with open-loop control.

1.7 MATLAB and Simulink

```
% Initialization and model definition
init01;

% Discrete time system model. x = [lambda r p p_dot]'
delta_t = 0.25;                          % Sampling time
A1 = [0 1 0 0;
      0 0 -K_2 0;
      0 0 0 1;
      0 0 -K_1*K_pp -K_1*K_pd]*delta_t + eye(4);
B1 = [0;0;0;K_1*K_pp]*delta_t;

% Number of states and inputs
```

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

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

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

% Bounds
ul = -30*pi/180;                   % Lower bound on control
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;               % Last input to be zero
vub(N*mx+M*mu)  = 0;               % Last input to be zero

% Generate the matrix Q and the vector c
Q1 = zeros(mx,mx);
Q1(1,1) = 0.1;                     % Weight on state x1
Q1(2,2) = 0.1;                     % Weight on state x2
Q1(3,3) = 0.1;                     % Weight on state x3
Q1(4,4) = 0.1;                     % Weight on state x4
P1 = 1;                            % Weight on input
Q = gen_q(Q1,P1,N,M);              % Generate Q, hint: gen_q
c = zeros(N*mx + M*mu ,1);         % Linear constant term in the QP

% Generate system matrixes for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu);      % Generate A, hint: gen_aeq
beq = [A1*x0;0;0;0];               % Generate b

```

```
% Solve QP problem with linear model
tic
% Quadratic Minimization with Linear Constraints and Bounds:
[z,lambda] = quadprog(Q1,P1,[],[],Aeq,beq,vlb,vub);
t1=toc;

% Calculate objective value
phi1 = 0.0;
PhiOut = zeros(N*mx+M*mu,1);
for i=1:N*mx+M*mu
  phi1=phi1+Q(i,i)*z(i)*z(i);
  PhiOut(i) = phi1;
end

% Extract control inputs and states
u  = [z(N*mx+1:N*mx+M*mu);z(N*mx+M*mu)];

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];
```

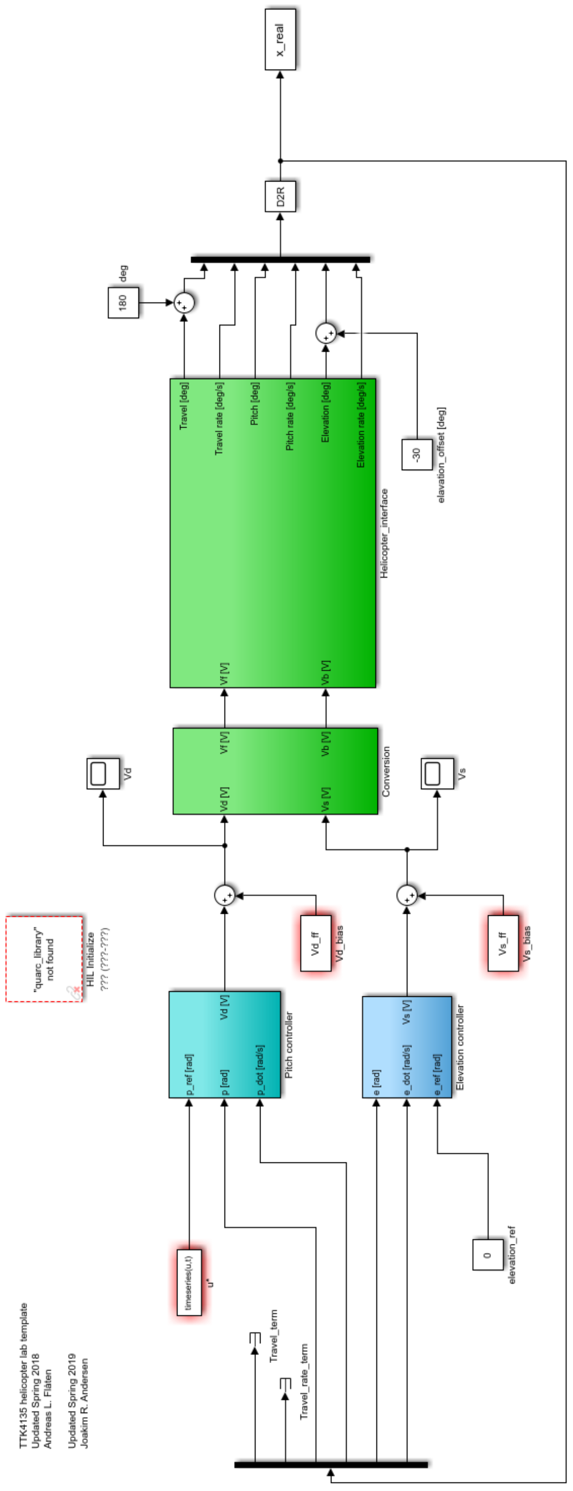


Figure 5: Simulink day 2.

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

2.1 LQ controller

Now we go from an open-loop to a closed-loop system with feedback, see fig. 6. The Linear Quadratic algorithm optimizes the $\boldsymbol{K}$ matrix with respect to cost. The controller aims to minimize the deviations from the optimal trajectory, given state and input penalization as described in the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$. The feedback is expressed as

$$u_k = {u_k}^* - \boldsymbol{K}^T(x_k - {x_k}^*) \tag{20}$$

$\boldsymbol{K}$ can be found by minimizing the cost function, given by

$$J = \int_0^\infty ({x_{i+1}}^T \boldsymbol{Q} \Delta x_{i+1} + \Delta {u_i}^T \boldsymbol{R} u_i) \tag{21}$$

for a linear model subject to the equality constraint

$$\Delta x_{i+1} = \boldsymbol{A} \Delta x_i + \boldsymbol{B} \Delta u_i. \tag{22}$$

In eq. (21) $\boldsymbol{Q}$ has to be positive and $\boldsymbol{R}$ has to be strictly positive. x and Δu is the deviation from the optimal trajectory. Defining the waiting matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ as 4x4 and 1x1 diagonal matrices respectively. $\boldsymbol{Q}$ represents the cost for state error, each element on the diagonal corresponding to an element in the x vector, λ, r, p and $\dot{p}$ respectively. $\boldsymbol{R}$ represents the cost for input. Each element on the diagonal corresponds to a specific element in u.

The $\boldsymbol{K}$ matrix is calculated in Matlab using the function `dlqr`:

```
[K, S, e] = dlqr(A1, B1, Q_lqr, R_lqr);
```

The dlqr function returns the optimal gain matrix $\boldsymbol{K}$, as well as $\boldsymbol{S}$, the solution of the discrete-time Riccati equation and $\boldsymbol{e}$; the eigenvalues of the closed-loop system $\boldsymbol{A_d} - \boldsymbol{B_d}\boldsymbol{K}$.

2.2 Model Predictive Control

An alternative strategy would be to use an Model Predivtive Control (MPC). This is realized by calculating a new solution to the LQ problem for each time step, in stead of for the entire time-horizon. The new solution is based on the solution from the previous timestep. One advantage of MPC is that it allows constraints. We ignored the inequality constraint $|p| \leq 30°$ when implementing the LQ-controller, but with MPC we can take this into consideration. MPC is also a more precise controller, resistant to disturbances. The LQ controller calculates one route from the initial starting point whereas the MPC calculates a new trajectory for each time step and therefore handles disturbances well. A disadvantage is that it takes more processing power than the LQ-controller. We also need feedback from all states. The structure of fig. 6 would look like fig. 7 if we used MPC.

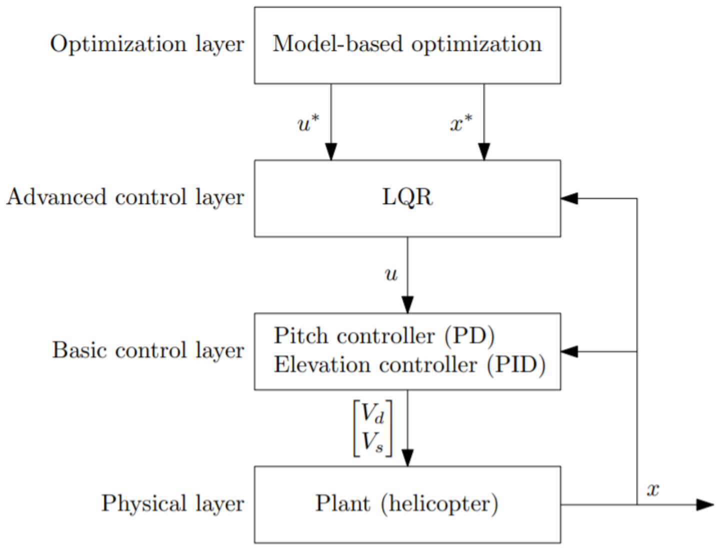


Figure 6: Illustration of the layers in the control hierarchy using LQR. Image: [1]

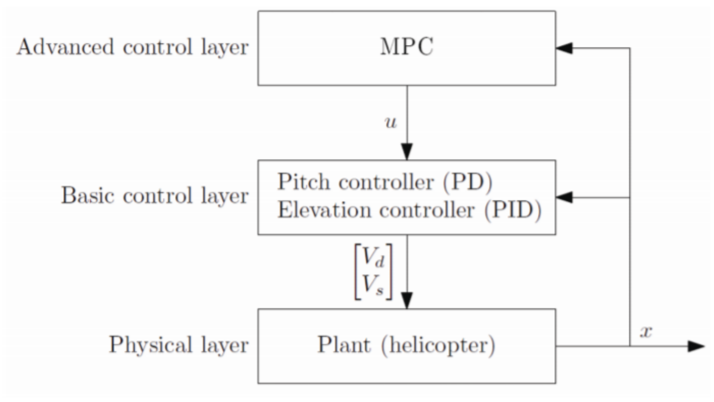


Figure 7: Illustration of the layers in the control hierarchy with MPC.

2.3 Experimental results

Starting out with

$$\boldsymbol{Q} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}, \boldsymbol{R} = 1 \tag{23}$$

Then we tried to increase R to 10. The results are shown in fig. 8 and fig. 9 respectively. We see that when we used $R = 1$ the deviation between the pre-calculated trajectory and and the measured trajectory for lambda is smaller than for $R = 10$. This is because we weight the input more. We found that $R = 1$ gave the least deviation and is thus the optimal R. Then we experimented with different values of

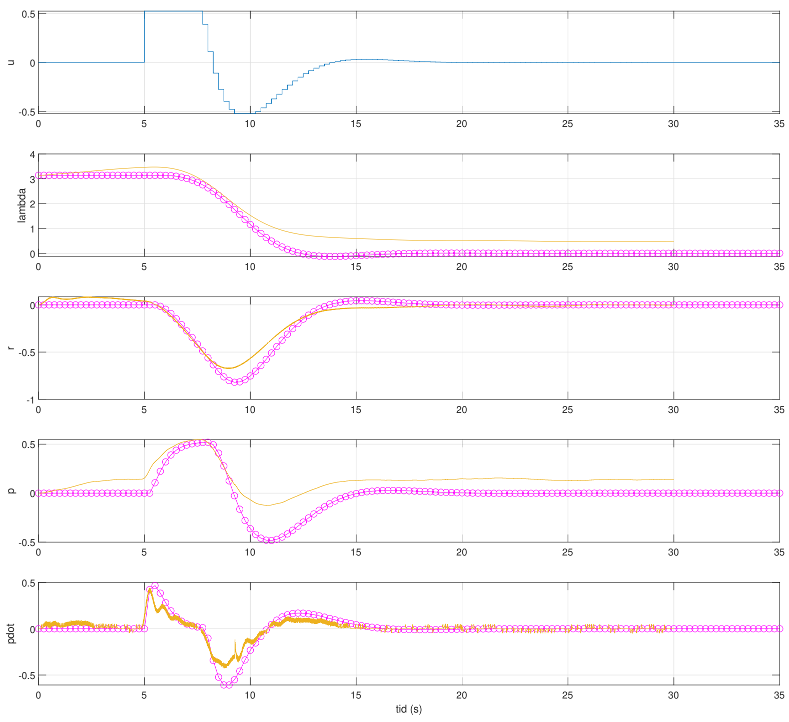


Figure 8: Plot of result using eq. (23).

$\boldsymbol{Q}$, weighting the elements in x differently. We found that strict travel penalization gave the best result. Weighting the first element, λ, the most because the helicopter tended to overshoot the final value λ_f. The penalization on the pitch and pitch rate should be low, on the other hand. This allows deviations from the pre-calculated trajectory.

After some experimentation we found that

$$\boldsymbol{Q} = \begin{bmatrix} 50 & & & \\ & 10 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \boldsymbol{R} = 1 \tag{24}$$

gave the best response, see fig. 10.

Compared to the previous task the travel has a constant deviation, not one that grows with time. There are no offset in travel rate, but an offset in pitch that we did not see earlier. This is due to the helicopter knowing when to stop and calculating that the trajectory is too aggressive. Yet, there is a significant improvement in

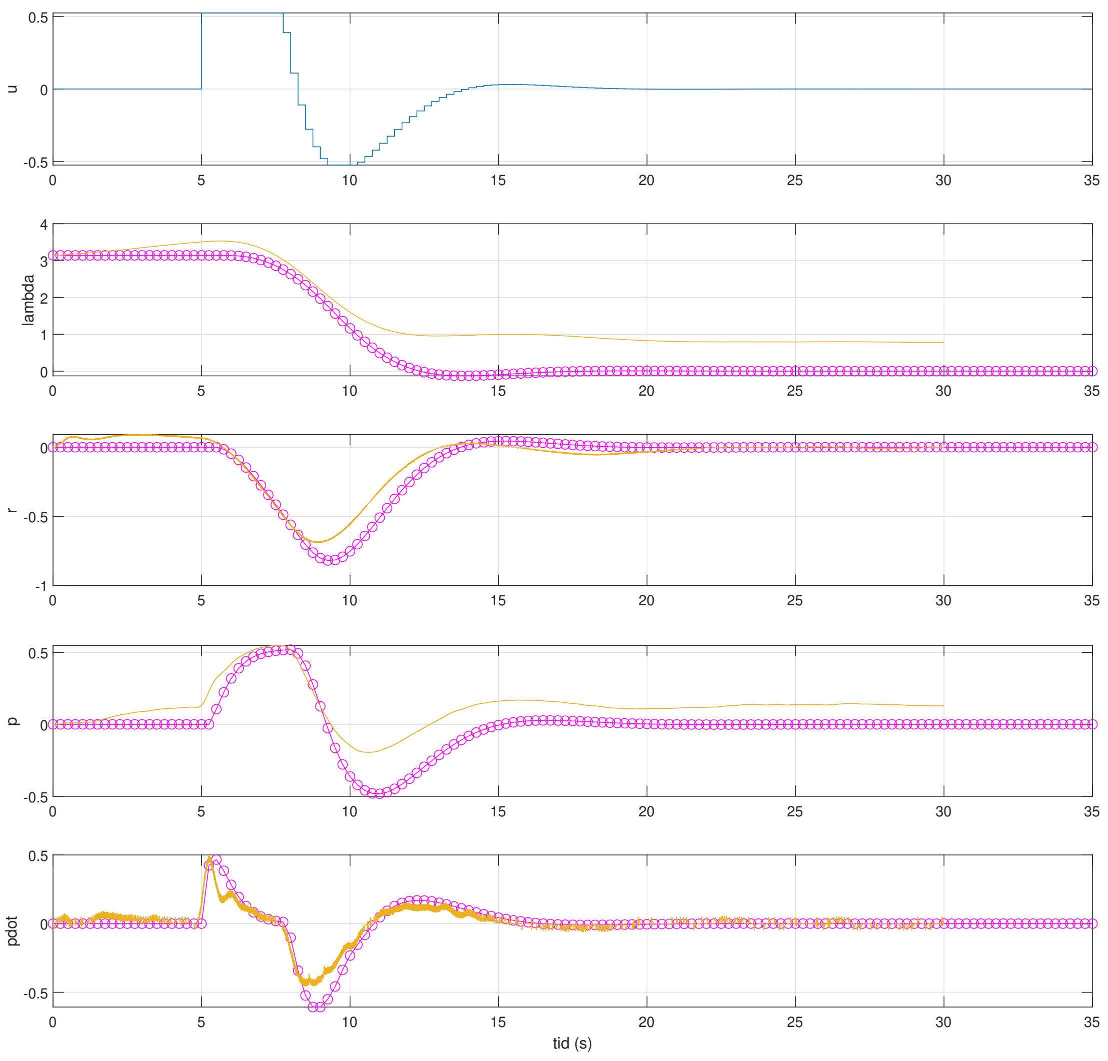


Figure 9: Plot of result using eq. (23) only with R = 10.

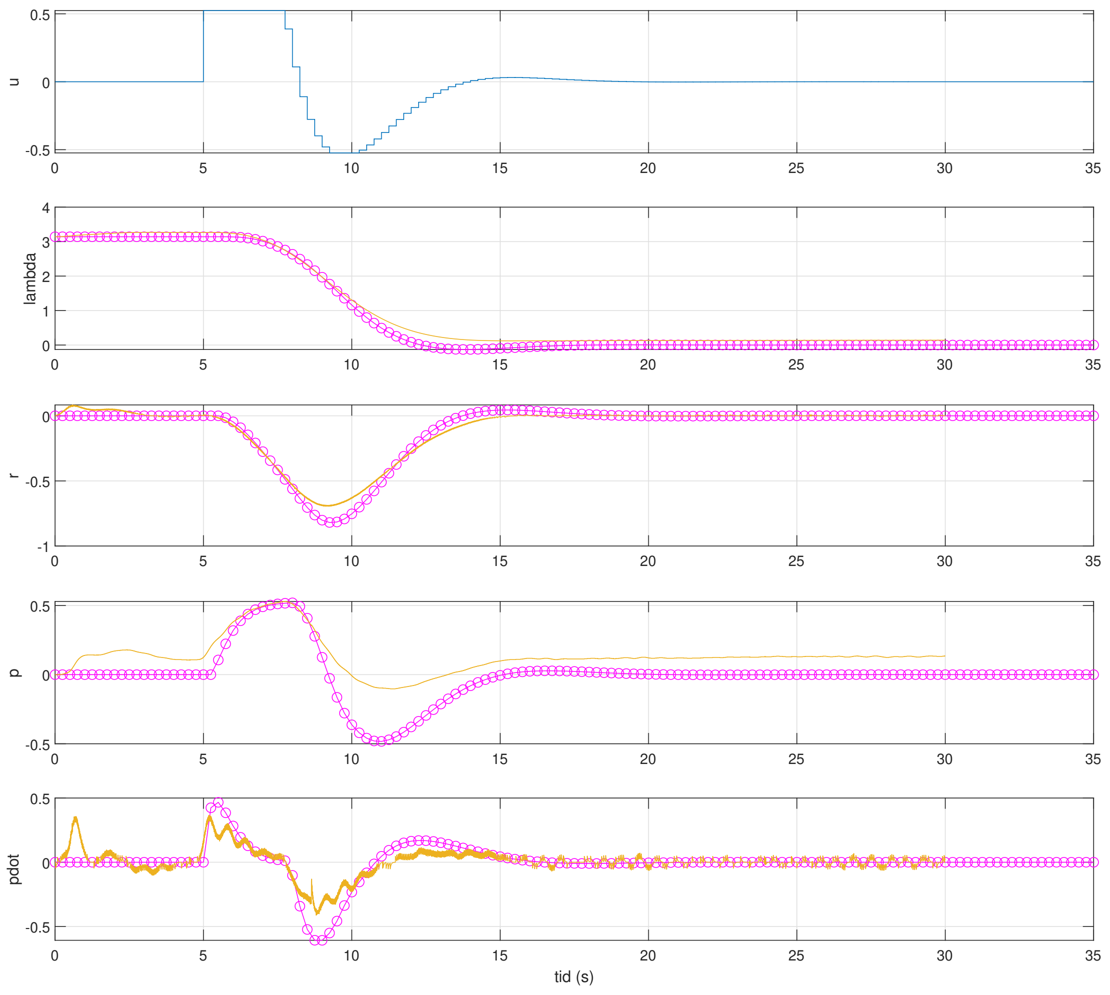


Figure 10: Optimal Q and R, using eq. (24).

controlling the system from fig. 2b to fig. 10. After adding the feedback by using an LQ controller we have made the system more stable and eliminated the deviation in travel.

2.4 MATLAB and Simulink

```
% Initialization and model definition
init01;

```

```
% Discrete time system model: x = [lambda r p p_dot]'
delta_t = 0.25;    % sampling time

A1 =   [0, 1, 0, 0;
        0, 0, -K_2, 0;
        0, 0, 0, 1;
        0, 0, -K_1*K_pp, -K_1*K_pd]*delta_t + eye(4);

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

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

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

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

% 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
Q1 = zeros(mx,mx);
```

```
Q1(1,1) = 2;                    % 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 = 2;                         % Weight on input
Q = gen_q(Q1,P1,N,M);           % Generate Q, hint: gen_q
c = zeros(n, 1);                % Generate c

% Generate system matrixes for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu); % Generate A, hint: gen_aeq
beq = Aeq*z;                    % Generate b
beq(1) = x1_0;

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

% Calculate objective value
phi1 = 0.0;
PhiOut = zeros(N*mx+M*mu,1);
for i=1:N*mx+M*mu
  phi1=phi1+Q(i,i)*z(i)*z(i);
  PhiOut(i) = phi1;
end

% Extract control inputs and states
u  = [z(N*mx+1:N*mx+M*mu);z(N*mx+M*mu)] % Control input from solution

x1 = [x0(1);z(1:mx:N*mx)];              % State x1 from solution
x2 = [x0(2);z(2:mx:N*mx)];              % State x2 from solution
x3 = [x0(3);z(3:mx:N*mx)];              % State x3 from solution
x4 = [x0(4);z(4:mx:N*mx)];              % State x4 from solution

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

u   = [zero_padding; u; zero_padding];
x1  = [pi*unit_padding; x1; zero_padding];
x2  = [zero_padding; x2; zero_padding];
x3  = [zero_padding; x3; zero_padding];
x4  = [zero_padding; x4; zero_padding];

```

```
t = 0:delta_t:delta_t*(length(u)-1);

Q_lqr = diag([50; 10; 1; 1]);
R_lqr = diag([10]);

[K, S, e] = dlqr(A1, B1, Q_lqr, R_lqr);

ts_x1 = timeseries(x1, t);
ts_x2 = timeseries(x2, t);
ts_x3 = timeseries(x3, t);
ts_x4 = timeseries(x4, t);

ts_u = timeseries(u, t);

ts_x_opt = timeseries([x1 x2 x3 x4], t);
```

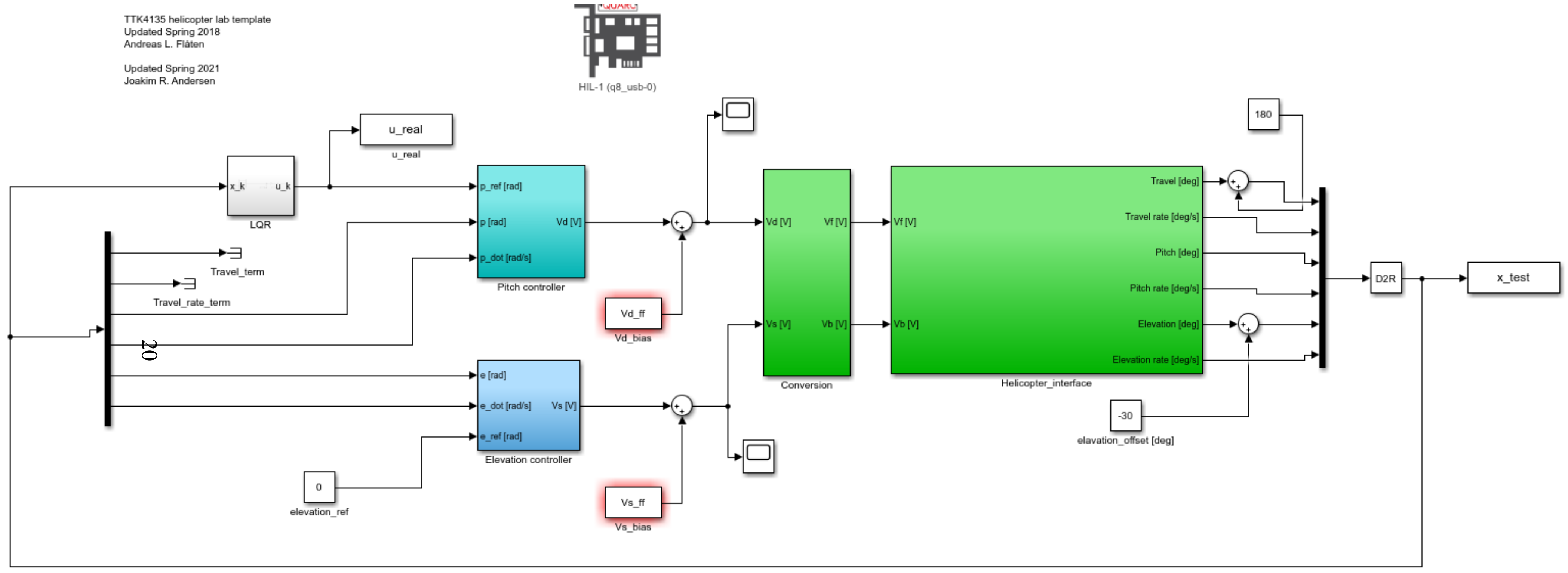


Figure 11: Simulink day 3.

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

3.1 The continuous model

In this task we want to write the system on continuous state space form using the equation in eq. (1). However, to achieve the goal of navigating the helicopter around an obstacle in the elevation direction, two extra states, e and $\dot{e}$, are added. This yields two extra input states for controlling these, x and u is now defined as

$$x = \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \\ e \\ \dot{e} \end{bmatrix}, \; u = \begin{bmatrix} p_c \\ e_c \end{bmatrix}. \tag{25}$$

This yields the state space form, using eq. (3a).

$$\boldsymbol{A}_c = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3K_{ep} & -K_3K_{ed} \end{bmatrix}, \boldsymbol{B}_c = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3 \end{bmatrix} \tag{26}$$

3.2 The discretized model

The continuous state space model is discretized using the Forward Euler method. By using eq. (5) with the new $\boldsymbol{A}_\mathrm{c}$ and $\boldsymbol{B}_\mathrm{c}$ the discrete time system yields

$$\boldsymbol{A}_\mathrm{d} = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & -\Delta t K_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & -\Delta t K_1K_{pp} & 1-\Delta t K_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & -\Delta t K_3K_{ep} & 1-\Delta t K_3K_{ed} \end{bmatrix} \quad \boldsymbol{B}_\mathrm{d} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \Delta t K_1K_{pp} & 0 \\ 0 & 0 \\ 0 & \Delta t K_3K_{ep} \end{bmatrix} \tag{27}$$

3.3 Experimental results

The results are shown in fig. 12. We see that the measurement of the helicopter does not follow the trajectory particularly well. Figure 13 shows the trajectory after tuning the $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices,

$$\boldsymbol{Q} = \begin{bmatrix} 5 & & & & & \\ & 0.2 & & & & \\ & & 2 & & & \\ & & & 0.2 & & \\ & & & & 1.5 & \\ & & & & & 0.2 \end{bmatrix}, \boldsymbol{R} = \begin{bmatrix} 1 & \\ & 1 \end{bmatrix} \quad . \tag{28}$$

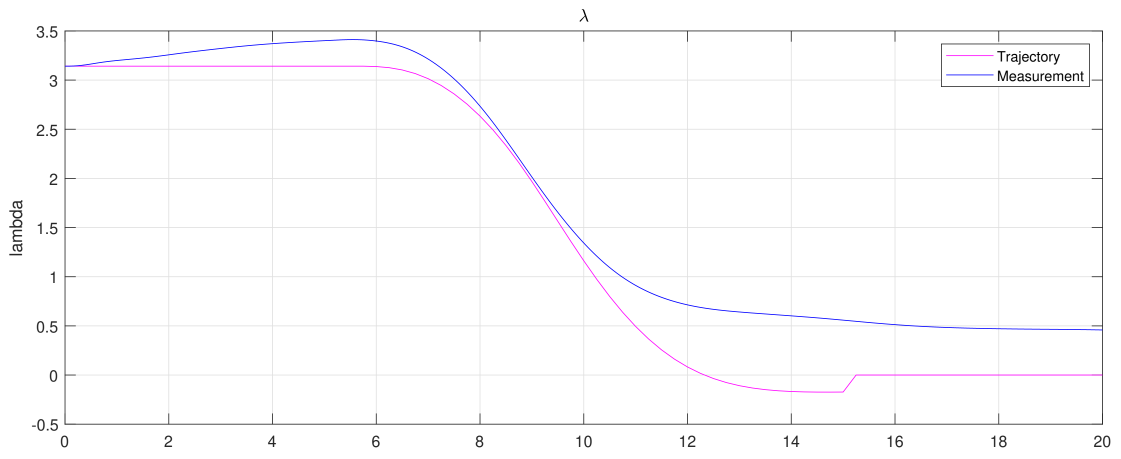
These weights are chosen with the goal of the task in mind: to pivot the helicopter 180°and respecting the constraint in the elevation. After considering how each element on the diagonal corresponds to the elements in the x vector. The optimal $\boldsymbol{Q}$ weights the travel angle the most, and travel rate, pitch rate and elevation rate the least. The elements in $\boldsymbol{R}$ weights the elements in u equally.

From fig. 13 we see that the offset in travel angle is smaller than in fig. 12. The elevation does not respect the constraint any more than before, and does still not follow the trajectory particularly well, but due to the improvement in travel angle we consider this tuning to be better. The reason the helicopter is not able to fly higher than shown in the plots, is because there is a deviation in elevation. The helicopter compensates by increasing the voltage sum, V_s and not by changing the pitch - resulting in the helicopter flying faster horizontally, not higher vertically. This behaviour is due to decoupling, which is discussed next.

3.4 Decoupled model

The first four states in the model, λ, r, p, and $\dot{p}$ are decoupled from the last two, e and $\dot{e}$. However, in reality all three states; travel, pitch and elevation, is interleaved. For example, we need to consider pitch if we want the helicopter to reach a certain elevation angle. This results in the elevation not being able to fly above the constraints, as seen in fig. 12. Ideally, the pitch should decrease when elevation is supposed to increase.

To avoid this problem, one could ideally have a more accurate and realistic model of the system where the pitch and elevation are interconnected. This would, however, result in a nonlinear system which is more complex to work with, and also not a part of the TTK4135 course. Another way to achieve the helicopter to fly higher, or ideally over the constraint, is to add more constraints to the system. By changing the pitch constraint to a lower angle, the helicopter will theoretically be able to fly higher, independent of the pitch. Our attempt to modify the helicopter constraints is shown in section 3.5.



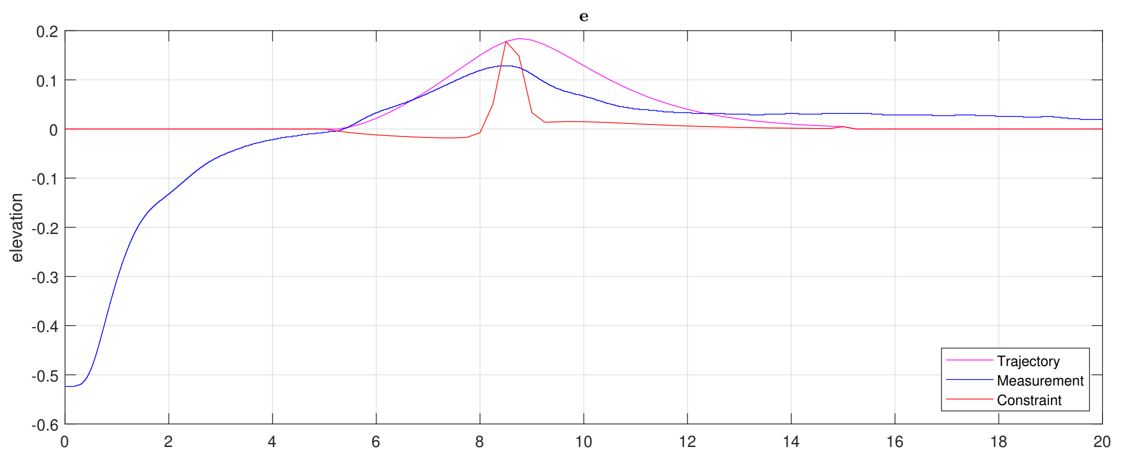


Figure 12: Travel and elevation with constraints.

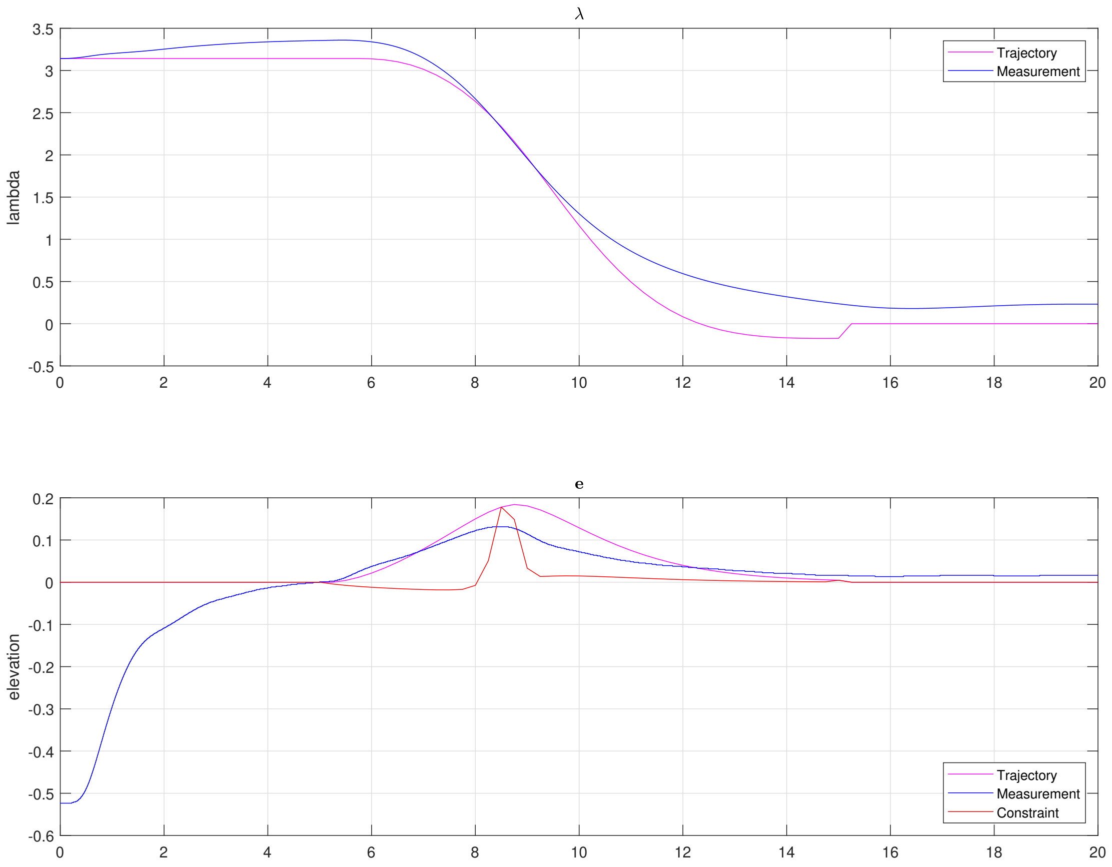


Figure 13: Travel and elevation with constraints after tuning.

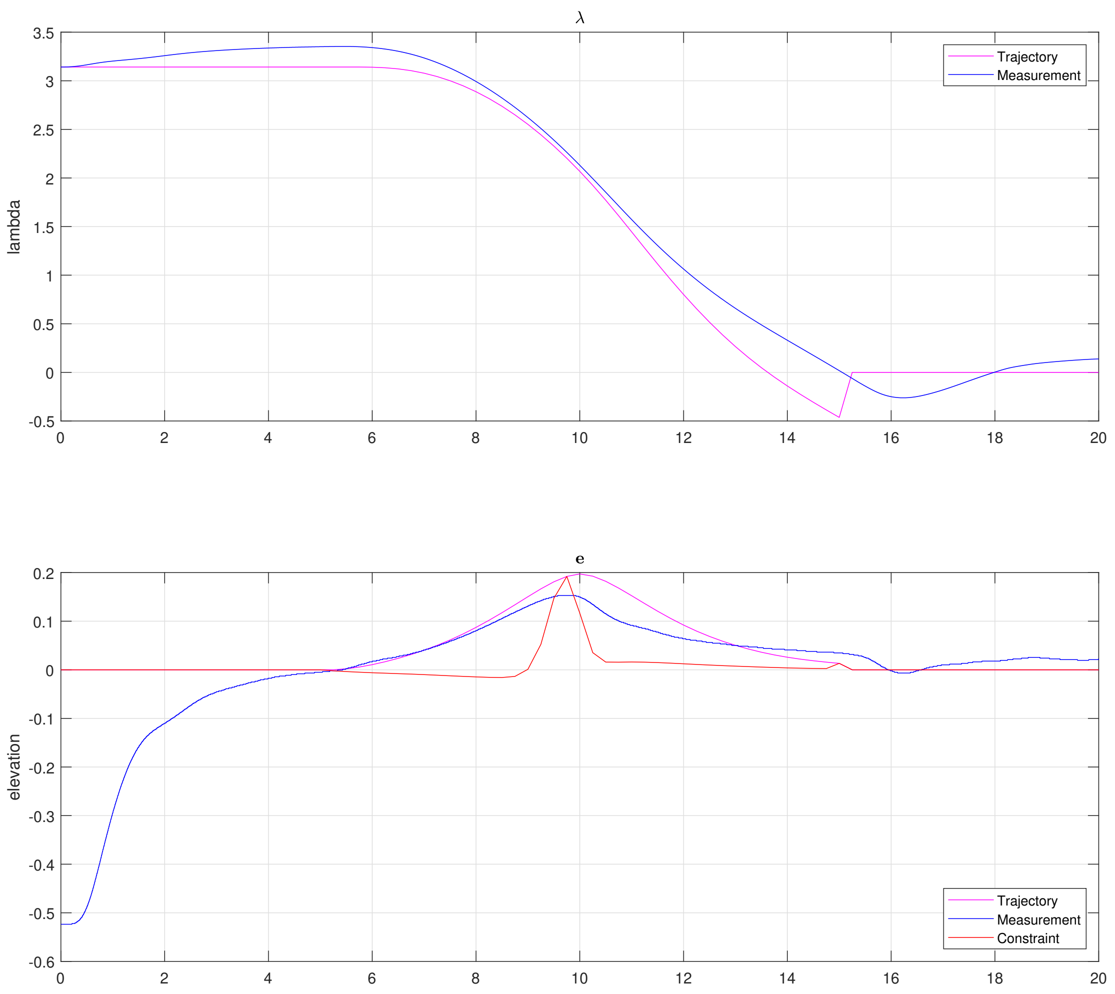


Figure 14: Added additional constraints.

3.5 Optional exercise

We added the constraint $|p_k| \leq \frac{15\pi}{180}$ on the pitch because this would help getting the helicopter respect the constraint in elevation angle more.

```
ul = [-15*pi/180 -Inf]    ;  % Lower  bound on control
uu = [15*pi/180  Inf]    ;   % Upper  bound on control
```

The result is shown in fig. 14. Compared to fig. 13 the helicopter is now closer to the trajectory. Yet, it is not able to fully respect the constraint. This is due to the LQR trying to keep the deviation between the calculated and measured pitch as little as possible, which is unaffected by the elevation.

3.6 MATLAB and Simulink

```
% Initialization and model definition
init01;

% Discrete time system model. x = [lambda r p p_dot]'
delta_t = 0.25; % sampling time
A1 = [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]*delta_t + eye(6);
B1 = [0 0 0 K_1*K_pp 0 0;
      0 0 0 0 0 K_3*K_ep]'*delta_t;

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

% Initial values
x1_0 = pi;                 % Lambda
x2_0 = 0;                  % r
x3_0 = 0;                  % p
x4_0 = 0;                  % p_dot
x5_0 = 0;
x6_0 = 0;
x0 = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_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

% Bounds
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)    = ul(1);            % Lower bound on state x3
xu(3)    = uu(1);            % 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
Q1 = zeros(mx,mx);
Q1(1,1) = 1;                 % Weight on state x1
P1 = diag([5 5]);            % Weight on input
Q = 2*gen_q(Q1,P1,N,M);      % Generate Q, hint: gen_q

% Generate system matrices for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu); % Generate A, hint: gen_aeq
beq = zeros(N*(mx),1);        % Generate b
beq(1:mx) = A1*x0;

% Solve QP problem with linear model
fun = @(z) 0.5*z'*Q*z;
nlc = @nonlincon;
options=optimoptions('fmincon','MaxFunEvals',Inf, 'MaxIter', 60000);
tic
[z, fval]=fmincon(fun, z0, [], [], Aeq, beq, vlb, vub, nlc, options);
t1=toc;

% Calculate objective value
phi1 = 0.0;
PhiOut = zeros(N*mx+M*mu,1);
for i=1:N*mx+M*mu
  phi1=phi1+Q(i,i)*z(i)*z(i);
  PhiOut(i) = phi1;
end

% Extract control inputs and states
u1 = [0; z(N*mx +1:2:N*mx + M*mu)];      % Control input from solution
u2 = [0; z(N*mx +2:2:N*mx + M*mu)];
x1 = [x0(1);z(1:mx:N*mx)];               % State x1 from solution
x2 = [x0(2);z(2:mx:N*mx)];               % State x2 from solution
x3 = [x0(3);z(3:mx:N*mx)];               % State x3 from solution
x4 = [x0(4);z(4:mx:N*mx)];               % State x4 from solution
x5 = [x0(5);z(5:mx:N*mx)];
x6 = [x0(6);z(6:mx:N*mx)];
```

```

num_variables = 5/delta_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];

u = [u1 u2];

t = 0:delta_t:delta_t*(length(u)-1);

% LQR 6 states
Q_LQR = diag([5 0.2 2 0.2 1.5 0.2]);
R_LQR = diag([1 1]');
[K, S, e] = dlqr(A1, B1, Q_LQR, R_LQR);

[c, ceq] = nonlincon(z);
c = [zero_padding;[c;0]; zero_padding];
c = c + x5;
c = timeseries(c,t);
```

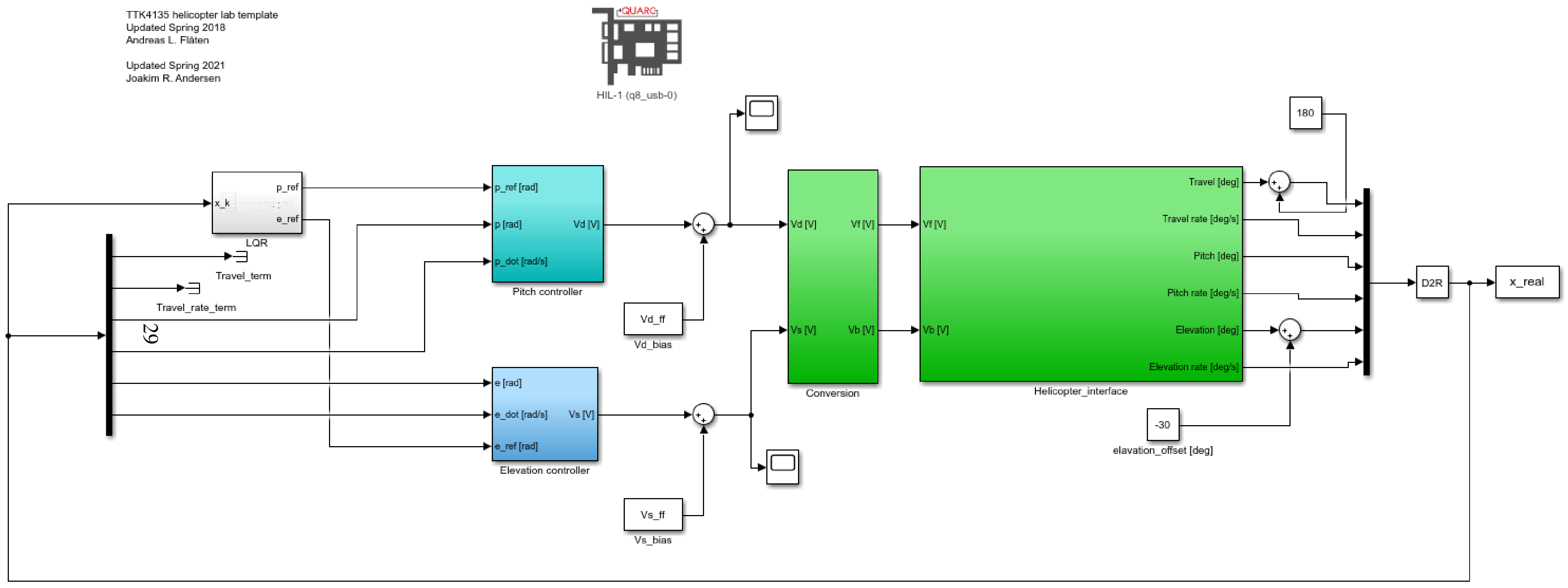


Figure 15: Simulink day 4.

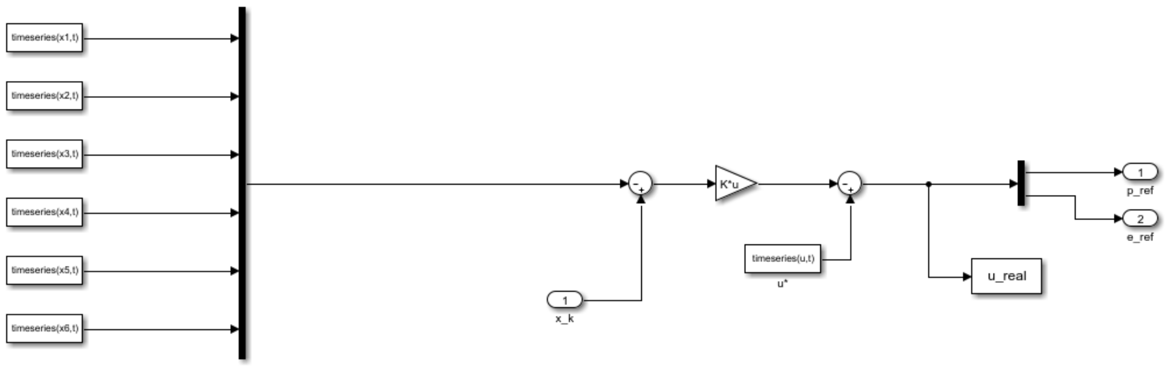


Figure 16: LQR day 4.

References

[1] *Lab exercise*. https://ntnu.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_16263_1&content_id=_674980_1&mode=reset. Accessed: 2021-03-20.