TTK4135 Optimization and Control
Lab Report

736698, 742388, 742367, 755875 and 731918
Group 17

May 1, 2015

Department of Engineering Cybernetics
Norwegian University of Science and Technology

Abstract

The purpose of the helicopter project in TTK4135 is to get experience combining optimization algorithms and control theory on a real world platform. The helicopter will be controlled using an optimal trajectory computed from a linearized model, both in open-loop and closed-loop with a linear quadratic regulator. The project also gives an introduction to hardware-in-the-loop (HIL) and automatic generation of code using MATLAB and QuaRC.

The main topics of this project are:

- Formulating and discretizing a state-space model for the helicopter
- Optimal control with linear inequality constraints
- Applying linear quadratic regulator (LQR) to the optimal trajectory
- Optimal control in two dimensions with non-linear inequality constraints

Contents

1 Intro

1.1 Modelling and linearization

In this lab we will be using optimization theory to compute an optimal trajectory, with a corresponding input, for a small helicopter. We do this by deriving a non-linear model for the system dynamics, and linearizing around an equilibrium. This method is widely used in control theory and works well if the system operates close to the linearization point. We linearize once about a fixed equilibrium, but a possibility is to redo the linearization as new measurements of the system are made.

1.2 Optimal control, limitations without feedback

Optimization theory is used in many applications, from economy and production planning to control theory. In this lab we will be looking at the last application, and how it can be used to solve control theory problems. The main difference between optimal control and a conventional error-feedback controller, is the ability to "look into the future", as opposed to only using the current state.

While a conventional regulator will calculate an error and correct the input thereafter, a planned trajectory from an optimization problem will contain inputs for the complete horizon. However, straightforward application of the optimal input sequence will rarely give good results, as modelling errors will cause the system to deviate from the optimal trajectory.

1.3 Optimal control with feedback

To compensate for modelling errors, it is useful to include some form of output feedback to correct the system towards to the planned trajectory. In this lab we use a linear-quadratic regulator, which minimizes a quadratic criteria given by tuning-friendly weight matrices. As long as the system is following the calculated planned trajectory, the original input sequence is used. If the system deviates from the trajectory, the input will be modified by a feedback term.

1.4 Optimal control with non-linear constraints, and thoughts on MPC

A common application of optimal control is to implement constraints on the system state or input. For instance, to avoid an object or limit the usage of motor thrust. Such constraints can be non-linear, which will significantly increase the time it takes to solve the optimization problem. This is not a concern if the trajectory is only computed once before running the system.

However when using MPC, which recomputes the horizon at each time step as measurements are made, the computation time becomes a problem. While state of the art solvers show promising results, they are currently difficult to implement on restrictive hardware, such as low-power microcontrollers.

2 Problem Description

2.1 Hardware setup

The helicopter is fixed to a base and has an extended arm with motors attached at one end, and a balance weight at the other. We parametrize the helicopter state by three rotations:

- Travel (λ): At the base about the vertical axis
- Elevation (e): At the arm joint about a horizontal axis
- Pitch (p): At the head joint about the arm axis

These angles are measured by the encoders at each rotational joint. We can affect the motion by adjusting the voltage input to the DC motors. Propellers are attached to the motors, and the thrust generated is assumed to be proportional to the voltage applied, with identical motor constant for both motors. Figure (1) shows the helicopter and the relevant angles. Table (1) shows the parameters for the helicopter used in the report.

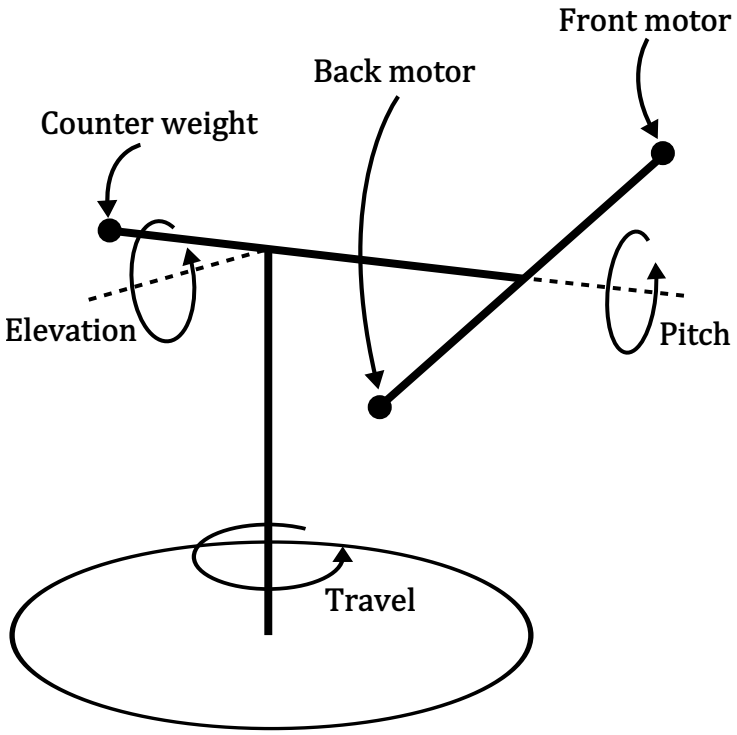

Figure 1: The helicopter hardware setup

2.2 Software setup

We use Simulink and Matlab to implement the control program, and interface with the helicopter through QuaRC and Realtime Workshop. See [Jesmani (2015)] for more information.

2.3 Model discussion

In order to compute an optimal control input, we need some measure of optimality. To this extent, we use a discrete-time state-space model, that describes the motion of our helicopter given an input sequence. The first step is to derive the equations of motions in continuous-time. We do this by considering torque balances about each rotation axis, and applying Newton's law. A full derivation is not

interesting, so we refer to the exercise text [Jesmani (2015)] and list the resulting set of differential equations below.

$$\ddot{e} + K_3 K_{ed} \dot{e} + K_3 K_{ep} e = K_3 K_{ep} e_c \tag{1a}$$

$$\ddot{p} + K_1 K_{pd} \dot{p} + K_1 K_{pp} p = K_1 K_{pp} p_c \tag{1b}$$

$$\dot{\lambda} = r \tag{1c}$$

$$\dot{r} = -K_2 p \tag{1d}$$

The associated nomenclature is listed in table (2). This model deserves some discussion. First, a complete model would have (non-linear) trigonometric terms. But such a model is impractical for use in dynamic optimization schemes. Instead, we linearize the model by assuming that the pitch and elevation angles are both close to zero.

The resulting model has the advantage of being compatible with highly efficient techniques that have been developed specifically for linear optimization problems. It is however a very simple model, and omits any interaction between elevation and pitch angle/travel. While there will always be modelling errors due to process noise, or inaccurate measurements/parameters, this simplification will affect the computed input. We can dampen the effect of the error by keeping the system close to the linearization point, for instance by constraining the angles to be within a margin around equilibrium.

Second, the model incorporates a PID controller for the elevation, which is assumed to counteract the constant torque due to gravity, as well as a PD controller for the pitch. The result of this is that our goal in the optimization problem is to compute the setpoints, e_c and p_c. Computing the appropriate voltages is left to the internal controllers.

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	0.20	N/V
J_e	Moment of inertia for elevation	1.11	$\mathrm{kg\,m^2}$
J_t	Moment of inertia for travel	1.11	$\mathrm{kg\,m^2}$
J_p	Moment of inertia for pitch	0.045	$\mathrm{kg\,m^2}$
m_h	Mass of helicopter	1.42	kg
m_w	Balance weight	1.80	kg
m_g	Effective mass of the helicopter	0.025	kg
K_p	Force to lift the helicopter from the ground	0.25	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
λ^*	Optimal travel trajectory
p_c^*	Optimal pitch setpoint
e_c^*	Optimal elevation setpoint

3 Repetition/Introduction to Simulink/QuaRC

This part is just a repetition, and therefore does not contain any important results. However, the part is important because it is used to test the hardware setup at the lab. This is done using a handed out model that made the helicopter hover at zero elevation and zero pitch. The helicopter suffered from minor drifting as a result of the motor being treated as equal, while they actually need to be tuned separately with one motor constant each. This was expected and did not introduce a problem since the drifting would be eliminated as soon as some form of feedback was implemented.

4 Optimal Control of Pitch and Travel without Feedback

In this section we compute an optimal trajectory x^* and a corresponding input sequence u^*. The trajectory should bring the helicopter from an initial state to another predefined state. We do not use feedback to correct for deviations from the calculated trajectory. Furthermore, we disregard elevation, that is, we assume $e = 0$. The computation of the optimal trajectory was formulated as a convex optimization problem.

4.1 The helicopter model on continuous-time state-space form

The system (1) describes the helicopter plant, with a basic control layer consisting of PID and PD controllers for elevation and pitch. The optimization layer gives the inputs to these regulators, as shown in figure (2).

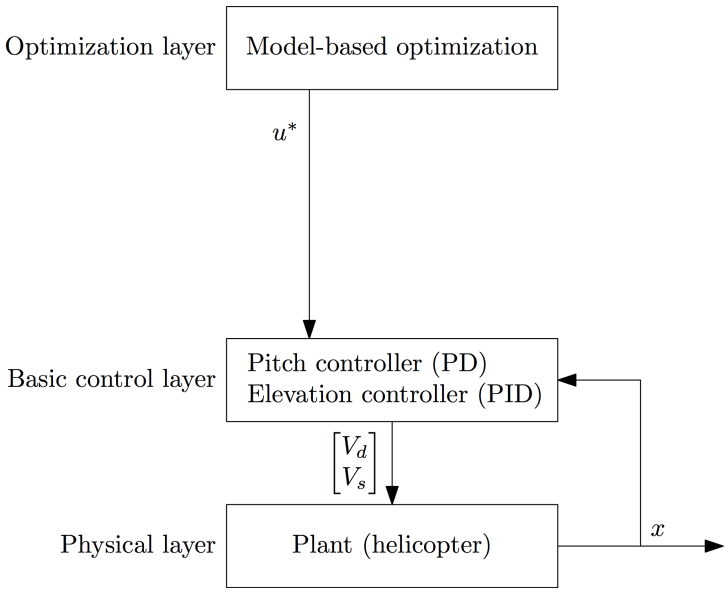

Figure 2: Control hierarchy

The system can be written on continuous-time state-space form:

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

with $x = \begin{bmatrix} \lambda & r & p & \dot{p} \end{bmatrix}^T$ and $u = p_c$. The continuous-time system matrices for this model are:

$$A_c = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} \end{bmatrix} \quad \text{and} \quad B_c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1 K_{pp} \end{bmatrix} \tag{3}$$

4.2 Discretization

The system dynamics was implemented as a sequence of constraints in the optimization problem, and the system model therefore had to be written in discrete-time state-space form,

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

The model was discretized using the Forward-Euler method with timestep h.

$$\dot{x}_k \approx \frac{x_{k+1} - x_k}{h} \tag{5}$$

Inserting this into (2),

$$\dot{x}_{k+1} \approx (I + hA_c)x_k + hB_c u_k \tag{6}$$

is obtained. A suitable approximation for the discrete-time matrices is then

$$A \approx I + hA_c \qquad \text{and} \qquad B \approx hB_c \tag{7}$$

These matrices were computed in Matlab, and are therefore not shown explicitly here.

4.3 Computation of optimal trajectory

An optimal trajectory can be generated by minimizing the cost function

$$\phi = \sum_{i=1}^{N} (\lambda_i - \lambda_f)^2 + q p_{ci}^2 \tag{8}$$

for some scalar weight $q \geq 0$ over the finite horizon of states and inputs

$$z = (x_1 \; x_2 \; \ldots \; x_N \; u_1 \; u_2 \; \ldots \; u_N)^T \tag{9}$$

This was done in Matlab using the function quadprog. The discrete system dynamics was implemented as equality constraints of the form $A_{\text{eq}} z = B_{\text{eq}}$, where A_{eq} and B_{eq} are given by the left- and right-hand side of the N constraints

$$\begin{aligned} x_1 - Bu_0 &= Ax_0 \\ x_2 - Ax_1 - Bu_1 &= 0 \\ &\vdots \\ x_N - Ax_{N-1} - Bu_{N-1} &= 0 \end{aligned}$$

We would also like to constrain the system state and input to be within a range

$$x^{\min} \leq x_{t+1} \leq x^{\max} \tag{10}$$

$$u^{\min} \leq u_t \leq u^{\max} \tag{11}$$

for $t = 0 \ldots N-1$. Applying these constraints to all states and inputs in the solution horizon, we have

$$\begin{bmatrix} I \\ -I \end{bmatrix} z \leq \begin{bmatrix} \{x_{t+1}^{max}\} \\ \{u_t^{max}\} \\ \{x_{t+1}^{min}\} \\ \{u_t^{min}\} \end{bmatrix}_{t=0..N-1} \tag{12}$$

,

which can be implemented as an inequality constraint of the form $A_{\text{iq}} z \leq B_{\text{iq}}$. Solving the optimization problem with different weights q did not lead to any significant differences in the trajectory, this because the model inaccuracies made the helicopter drift away from the desired state anyway.

The objective function (8) weights the input relative to the state deviations using the weight parameter q. The term $(\lambda_i - \lambda_f)^2$ is the state deviations. They are squared to prioritize the largest deviation. Note that the cost function (8) does not take into consideration that λ_i plus some multiple of 2π describes the same physical orientation of the helicopter. For example, if the reference is 0 and $\lambda_i = 2\pi$, it will be regarded as a large error, even though the helicopter is in fact in the desired orientation. A more optimal scheme would take this into consideration.

4.4 Implementation

Figure (3) shows the implementation with $q = 1$. Zeroes were added on both sides of the input sequence to give time to initialize and stabilize the helicopter. As can be seen from the figure, the helicopter does not reach its desired final state.

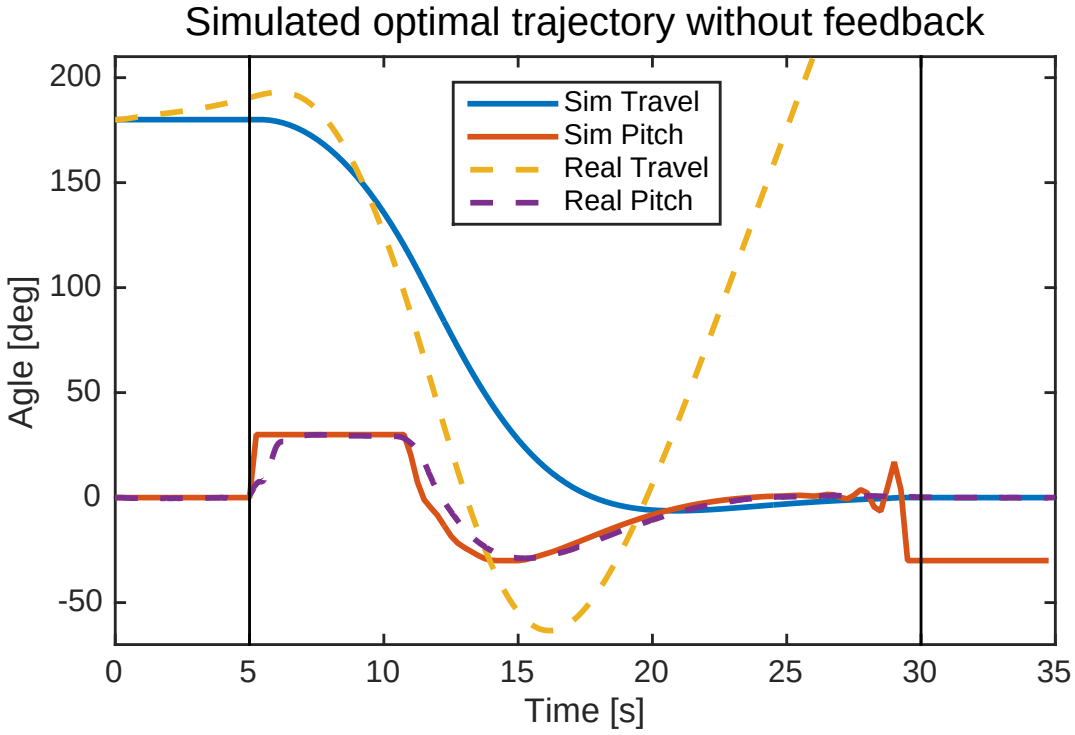

Figure 3: Plot of day 2

The deviation was caused by an imperfect model. A perfect model is unrealistic to construct, and without feedback, the model inaccuracies will lead to nonoptimal results in the real world. For example, the pitch-regulator in our model is fast enough to follow its input perfectly, and we are using a linear model that clearly does not correspond perfectly to the real helicopter.

5 Optimal Control of Pitch and Travel with Feedback (LQ)

This problem involves implementing an LQ controller for optimal control with feedback. We calculated a gain matrix K using the built in MATLAB function dlqr, which solves the discrete algebraic riccati equation. We also implemented feedback on the helicopter, and discussed if MPC is a good alternative to LQR.

5.1 Calculating the gain matrix K

Feedback for the system is introduced by using $u_k = u_k^* - K^T(x_k - x_k^*)$. Here x^* is the optimal trajectory and u^* is the optimal input sequence. A good choice for K can be found by minimizing the objective function:

$$J = \sum_{i=0}^{\infty} \Delta x_{i+1}^T Q \Delta x_{i+1} + u_i^T R u_i \tag{13}$$

Here, Q and R are user defined diagonal matrices chosen to weight deviations in states separately. The tuning of these matrices are discussed in the next section. The built in MATLAB function dlqr is used to solve the riccati equation for the corresponding optimization problem and to compute the gain matrix K.

5.2 Implementing feedback on the helicopter

The block diagram structure of the helicopter with feedback can be seen in figure (4).

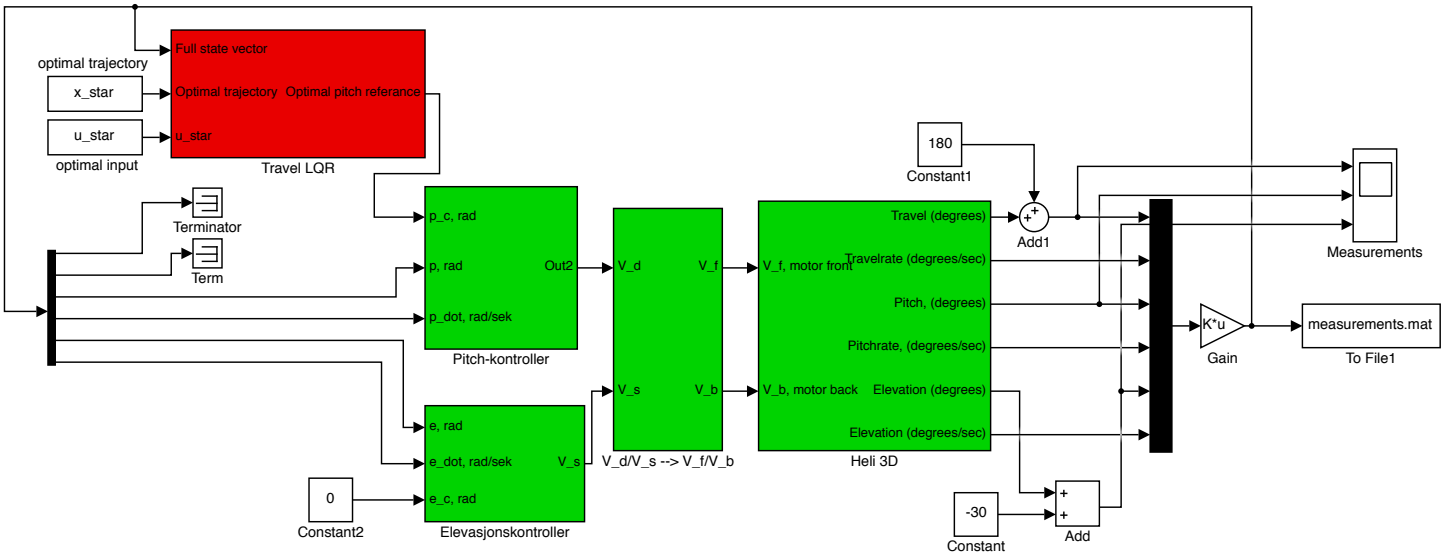

Figure 4: Simulink model with feedback

By keeping R constant at unity weight and tuning Q, we saw that by weighting travel more than pitch (50 1 1 1), the helicopter followed the travel trajectory closer than when weighting all states the same. Weighting deviation in pitch more than travel (1 1 10 1) gave a bad result. Figure (5) shows how the trajectory based on these different weights turned out compared to the mathematical optimal path. The reason for the bad trajectory tracking was that the optimal pitch reference trajectory was based on a linearized model, and also because of modelling error.

5.3 Comparison between LQR and MPC

The way to implement an MPC controller would be to calculate a new optimal trajectory at each time step. The first time step of each calculated optimal input sequence should then be used as the input u_t for that time step.

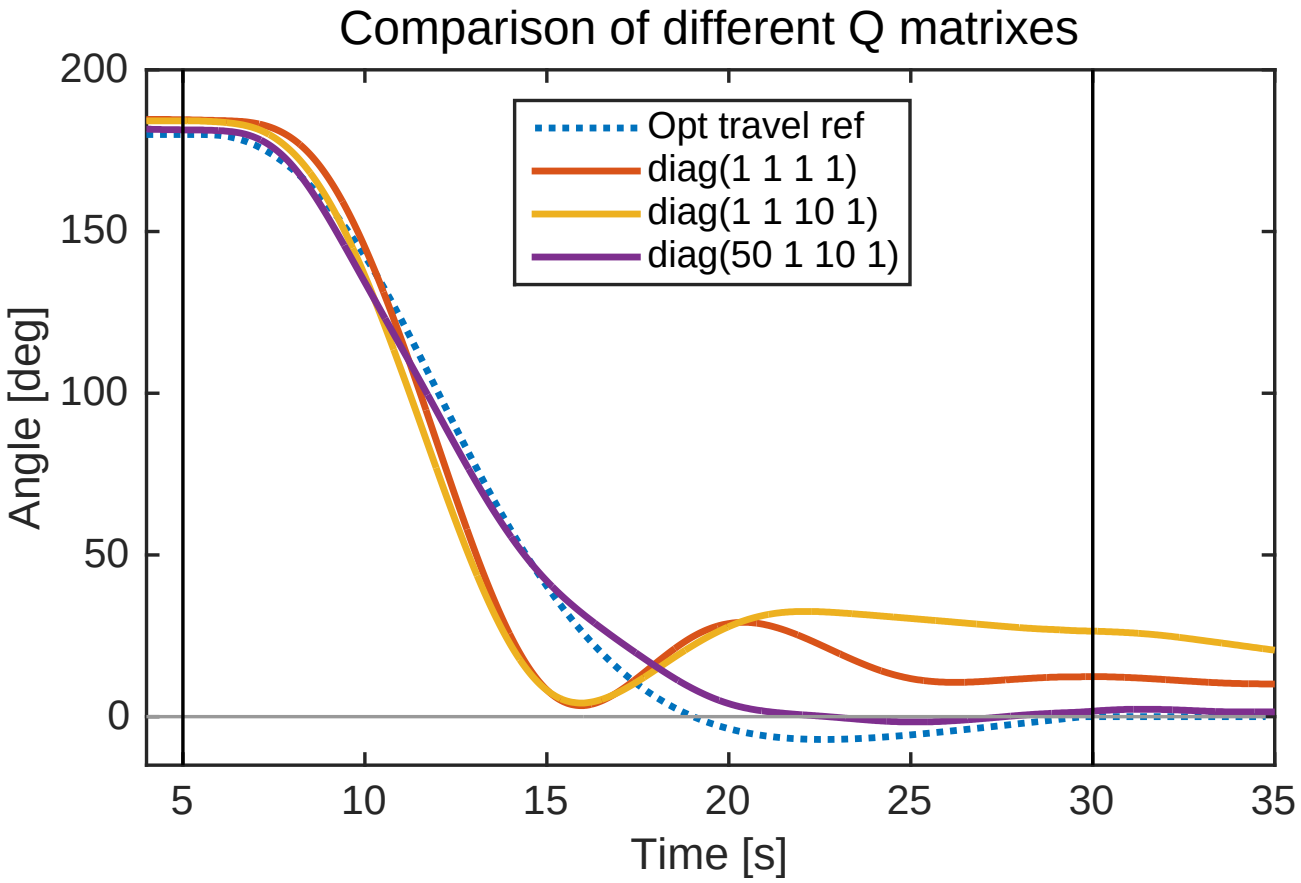

Figure 5: Travel trajectories with different weighting

The advantages of using MPC instead of LQR is that it gives the possibility to have constraints in the regulator. MPC might also result in a trajectory demanding less input, and provides implicit feedback. As we have already observed, feedback is a necessity for the control of a system with modelling errors and disturbances. The main disadvantage of using MPC is that the heavy calculations of finding an optimal trajectory would have to be processed during run time, hence it might not be able to control a system as fast as a helicopter. The control hierarchy with MPC would have replaced the Advanced control layer by the Optimization layer. Hence the optimal input sequence would be fed straight to the basic control layer.

6 Optimal Control of Pitch, Travel and Elevation with and without Feedback

In this section, we extended our model to include the remaining states, elevation e, and elevation rate $\dot{e}$. We used a non-linear solver to compute an optimal trajectory in two dimensions, and additionally constrained the elevation to avoid a restriction shaped as a bell-curve.

6.1 State-space formulation

The state-vector was extended with the remaining states,

$$x = \begin{bmatrix} \lambda & r & p & \dot{p} & e & \dot{e} \end{bmatrix}^T \tag{14}$$

and the elevation setpoint was added to the input-vector

$$u = \begin{bmatrix} p_c & e_c \end{bmatrix}^T . \tag{15}$$

The system is on the usual state-space form (2), with

$$\dot{x} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3K_{ep} & -K_3K_{ed} \end{bmatrix}}_{A_c} x + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3K_{ep} \end{bmatrix}}_{B_c} u \tag{16}$$

6.2 Discretization

We discretized (16) using the same method as in section (4). That is, an approximation of the discrete-time state-space matrices was

$$A \approx I + hA_c \qquad \text{and} \qquad B \approx hB_c \tag{17}$$

where I was the 6×6 identity matrix.

6.3 Modelling the restriction

A common application of optimal control is to implement restrictions, such as avoiding physical objects, as constraints in the optimization problem. Such restrictions can not be enforced when using only state-feedback controllers.

We wished to restrict the helicopter head to move above the bell-shaped curve

$$e_k \geq \alpha \exp(-\beta(\lambda_k - \lambda_t)^2) \tag{18}$$

for all timesteps k of the solution horizon. Since this is a non-linear constraint, we could no longer use a QP solver. Instead, a non-linear solver was needed, and in this case, the MATLAB function fmincon was used with a SQP-type algorithm.

6.4 Objective function

For this assignment, the cost function is the same as in equation (8), but with an extra term for penalizing elevation. We chose to set up the cost function on a more general form:

$$\phi = \sum (x^T Q x + u^T R u) \tag{19}$$

Here, Q and R are diagonal matrices, with unity weight on q_{travel}, r_{pitch} and $r_{elevation}$, so that it corresponds with the extended state and input vectors (14).

6.5 Results

For controlling the helicopter, we ended on a horizon of $N = 60$ timesteps, or 15 seconds. This gave an optimal trajectory that looked much the same as for assignment 3 for pitch and travel. In addition, an elevator trajectory just touching the top of the bell curve constraint was also calculated. The suggested horizon of $N = 40$ did not terminate in reasonable time, and was therefore replaced by a longer horizon giving reasonable results.

The solution without feedback acted in the same manner as it did in section (4). This can be seen in figure (6), the real trajectory of travel and elevation tried to follow the optimal solution, but the effect of not having a perfect model became more and more clear as time passed, and at the end of the horizon, the travel drifted away from the solution just like in section (4).

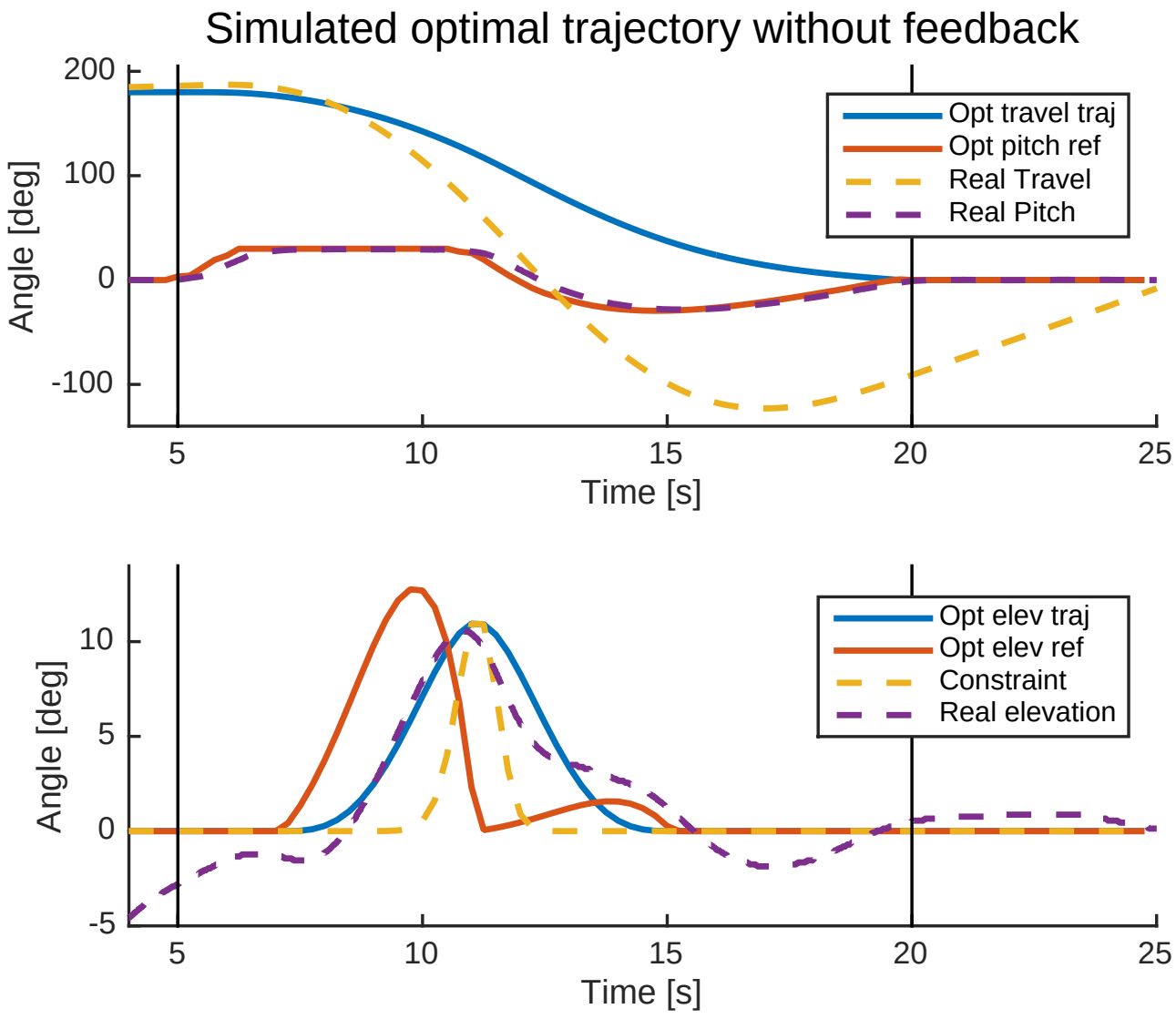

Figure 6: Problem 4 open loop response

Using state feedback LQR, the helicopter managed to follow the calculated trajectory quite good after some tuning. As in section (5), it was appropriate to weight deviation(s) in travel (and here also elevation) more than deviation in pitch. When the regulator is working on an inaccurate model, it is for the task given more constructive to try to follow the travel and elevation trajectories rather than the ideal, linearized input trajectory. This input will, given the unlinearities of the model, not lead to

an appropriate response for the trajectories. The result, with different weights on the states of Q_{LQR} can be seen in the following figures.

One thing worth noticing is that the trajectory of elevation consequently fell below the constraint at the peak of the constraint. We believe this is due to the fact that our model suggests that there is no link between the model for elevation, and that of pitch. They are in fact linked, and because of this, the regulator doesn't manage to meet both demands perfectly at the same time, and we get a deviation. If the model would include these crossing terms, the result would probably be better. The problem with this approach is that it leads to a non linear objective function, which takes longer time to compute for the solver.

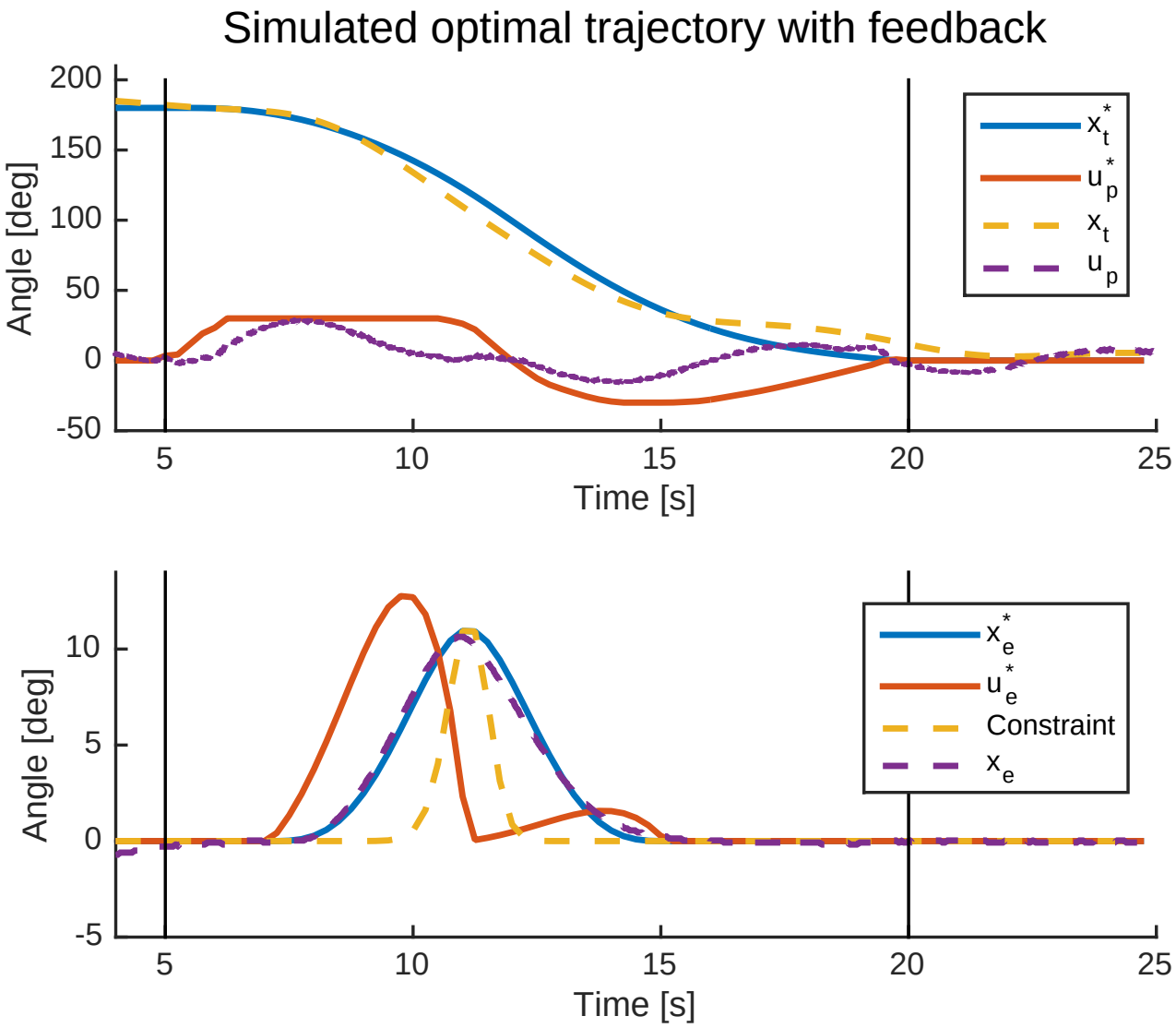

Figure 7: Plot day 4 closed loop Q=diag(20 1 1 1 30 10)

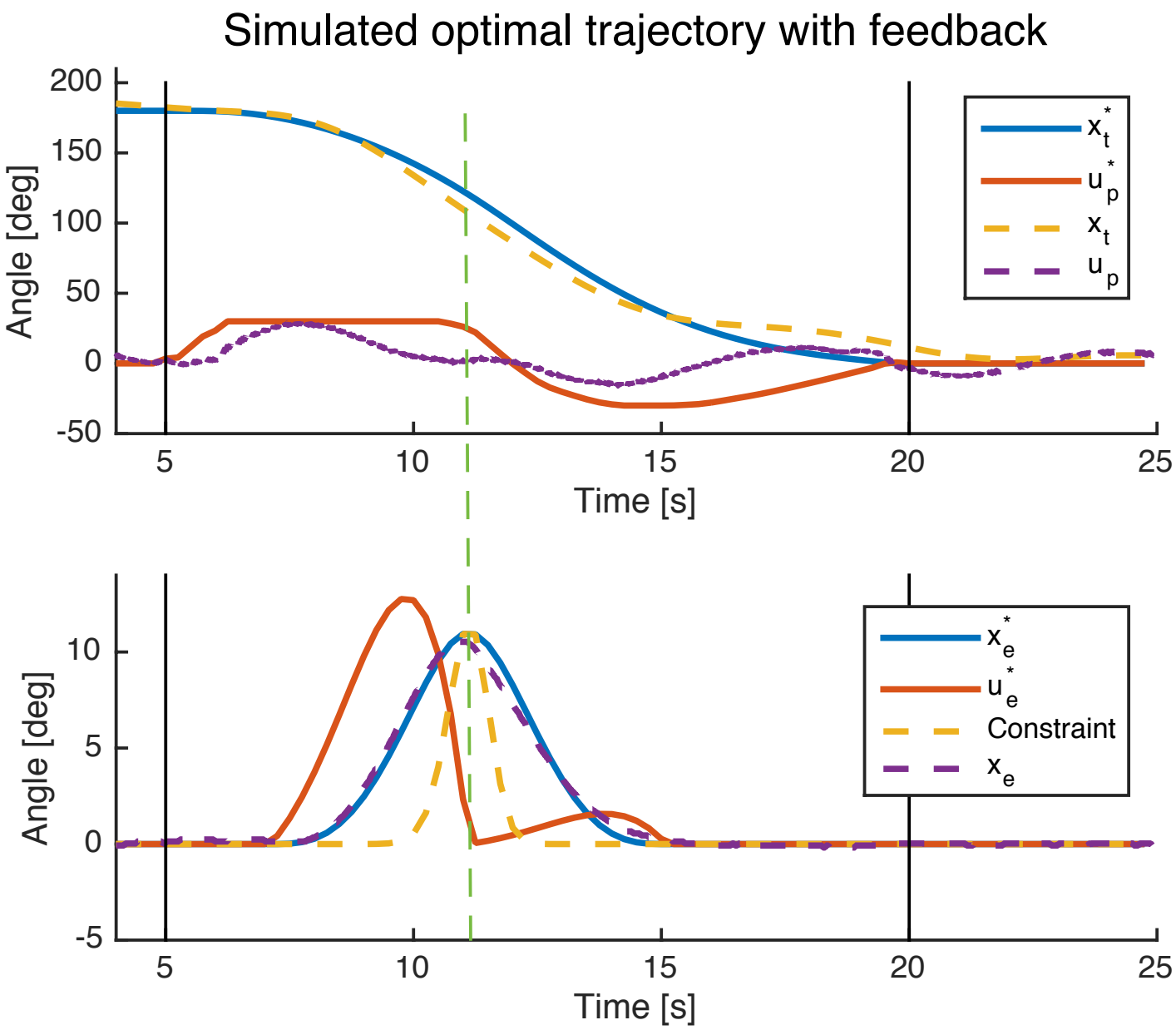

Figure 8: Coupling between pitch and elevation

7 Discussion

7.1 Optimal control without feedback

When controlling the helicopter with only the estimated optimal input sequence, the linearization and modeling errors becomes too large, and the helicopters behavior will deviate drastically from the estimated response. However, there is one benefit from using optimal control without feedback; the ability to limit inputs. So if the system need optimal control with limits on states and inputs, and also need feedback, MPC is the way to go. See section under.

7.2 Optimal control with feedback

While the open-loop configuration of optimal control was quite poor, the closed-loop configuration with a LQR was quite promising. Our testing showed that the optimal state trajectory was much more useful than the optimal input sequence. With a LQR this could be achieved by weighting the error in travel and pitch quite much. This way the regulator acts like a compensator on modeling errors since its additive with the optimal input sequence.

Another effect of including feedback on an optimal trajectory is that the final input is no longer guaranteed to be within the constraints of the optimization problem.

7.3 MPC with implicit feedback

As mentioned in the first section, optimal open-loop configuration performs quite poor caused by the lack of feedback. This is where MPC makes an entry, with its re-optimization every time step with the previous measured / estimated state as initial conditions. Since this is done every time step, only the first step of the input sequence is used. This way we get both an implicit feedback and the possibilities to set constraints on states and inputs.

8 Conclusion

Our results and discussion clearly points towards MPC as the most useful way of controlling a plant such as the helicopter model. However, it it also the most expensive algorithm in terms of computational power needed, and therefore might be considered as overkill in many small systems or systems with lower requirements on optimality.

A MATLAB Code

A.1 Day 2

```
run('../init.m'); %% Endret m_g til 0.025 fra 0.012
% Adjustable parameters
x0 = [pi 0 0 0]';         % Initial state
h  = 0.25;                % Discretization timestep
N  = 100;                 % Length of horizon
nx = 4;                   % Number of states of system
nu = 1;                   % Number of inputs of system
offsetTime = 5;           % Init time at start of simulation
n_offset = offsetTime/h; % Deadzone at start and end (timesteps)
Q = eye(nx);              % State penalty weights
Q(2,2) = 0;               % Free travel rate
Q(3,3) = 0.00;            % Free pitch
Q(4,4) = 0;               % Free pitch rate
R  = 1;                   % Input penalty weight
% System state and input bounds
pitch_lim = 30*pi/180;
x_max = [+inf +inf +pitch_lim +inf]';
x_min = [-inf -inf -pitch_lim -inf]';
u_max = +pitch_lim;
u_min = -pitch_lim;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n  = N * (nx + nu); % Length of manipulation variable vector
% Continuous-time system matrices
Ac = [0 1     0          0 ;
      0 0   -K_2         0 ;
      0 0     0          1 ;
      0 0 -K_1*K_pp -K_1*K_pd];

Bc = [0 ; 0 ; 0 ; K_1*K_pp];
% Discrete-time system matrices
A = eye(4) + h * Ac;
B = h * Bc;
% Generate equality constraints matrix
Aeq = gena2(A, B, N, nx, nu);
% Generate righthand side of equality constraints
Beq        = zeros(N*nx, 1);
Beq(1:nx) = A*x0;
% Generate quadratic objective matrix
H = genq2(Q, R, N, N, nu);
% Solve QP
[lb, ub] = genbegr2(N, N, x_min, x_max, u_min, u_max);
lb(nx*(N-1)+1) = 0; %Limit last state
ub(nx*(N-1)+1) = 0; %Limit last state
% lb((nx+nu)*N)  = 0; %Limit last input
% ub((nx+nu)*N)  = 0; %Limit last input
f = zeros(1, n);
z = quadprog(H, f, [], [], Aeq, Beq, lb, ub);
u = z(N*nx+1:n);
% Prepeare input sequence
t = (0:N+2*n_offset-1) * h;
pitch_input = zeros(N+2*n_offset, 2);
pitch_input(:, 1) = t;
pitch_input(n_offset+1:N+n_offset, 2) = u;

%% Plot simulated system
fig = figure(1); clf(1); box  on;
set(gca,'FontSize',11)
sim_travel = z(1:nx:N*nx);
sim_pitch  = z(3:nx:N*nx);
```

```
sim_travel_w_offset=[ones(n_offset,1)*pi;sim_travel;ones(n_offset,1)*sim_travel(end)];
sim_pitch_w_offset = [zeros(n_offset,1); sim_pitch ;ones(n_offset,1)*sim_pitch(end)];
time = (0:N-1+2*n_offset)*h;
hold all;
plot(time,(180/pi)*sim_travel_w_offset,'LineWidth',2);
plot(time,(180/pi)*sim_pitch_w_offset,'LineWidth',2);
legend('Sim Travel', 'Sim Pitch');

% Plot results
load ('measurements.mat');
t_real = measurements(1,:);
travel = (180/pi)*measurements(2,:);
pitch = (180/pi)*measurements(4,:);
plot(t_real,travel, 'LineWidth', 2,'LineStyle','--');
plot(t_real, pitch, 'LineWidth', 2, 'LineStyle', '--');
legend('Sim Travel', 'Sim Pitch','Real Travel', 'Real Pitch','Location','North');
title('Simulated optimal trajectory without feedback', 'FontSize',14,'FontWeight','normal');
xlim([0 (N*h)+2*offsetTime]); ylim([-70 210]);
line([offsetTime offsetTime],get(gca,'YLim'),'Color','Black','LineWidth',1);
line([offsetTime+N*h offsetTime+N*h],get(gca,'YLim'),'Color','Black','LineWidth',1);
xlabel('Time [s]'); ylabel('Agle [deg]');
set(fig, 'units','centimeters');
pos = get(gcf, 'position');
set(gcf, 'position', [pos(1), pos(2), 15, 9]);
```

A.2 Day 3

```
run('../init.m');
% Adjustable parameters
x0 = [pi 0 0 0]';         % Initial state
h  = 0.25;                % Discretization timestep
N  = 100;                 % Length of horizon
nx = 4;                   % Number of states of system
nu = 1;                   % Number of inputs of system
offsetTime = 5;           % Init time at start of simulation
n_offset = offsetTime/h; % Deadzone at start and end (timesteps)
Q = diag([1 0 0 0]);      % State penalty weights in optimization QP
R = diag([2]);            % Input cost in optimization QP
% System state and input bounds
pitch_lim = 30*pi/180;
pitch_rate_lim = 30*pi/180;
x_max = [+inf +inf +pitch_lim +pitch_rate_lim]';
x_min = [-inf -inf -pitch_lim -pitch_rate_lim]';
u_max = +pitch_lim;
u_min = -pitch_lim;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
n  = N * (nx + nu); % Length of manipulation variable vector
% Continuous-time system matrices
Ac = [0 1     0          0 ;
      0 0   -K_2         0 ;
      0 0     0          1 ;
      0 0 -K_1*K_pp -K_1*K_pd];

Bc = [0 ; 0 ; 0 ; K_1*K_pp];
% Discrete-time system matrices
A = eye(4) + h * Ac;
B = h * Bc;
% Generate equality constraints matrix
Aeq = gena2(A, B, N, nx, nu);
% Generate righthand side of equality constraints
Beq       = zeros(N*nx, 1);
Beq(1:nx) = A*x0;
% Generate quadratic objective matrix
H = genq2(Q, R, N, N, nu);
% Solve QP
[lb, ub] = genbegr2(N, N, x_min, x_max, u_min, u_max);
ub(nx*(N-1)+1) = 0*pi/180; % Limit last state
lb(nx*(N-1)+1) = 0*pi/180; % Limit last state
f = zeros(1, n);
z = quadprog(H, f, [], [], Aeq, Beq, lb, ub);
u = z(N*nx+1:n);
% LQR
Q_LQR = diag([1 1 10 1]);
R_LQR = eye(nu);
[K, S, E] = dlqr(A,B,Q_LQR,R_LQR);
% Create Simulink inputs
t = (0:N+2*n_offset-1) * h;
u_star = zeros(N+2*n_offset, 2);
u_star(:, 1) = t;
u_star(n_offset+1:N+n_offset, 2) = u;
x_star = zeros(N+2*n_offset, nx+1);
x_star(:, 1) = t;
x_star(1:n_offset, 2) = pi * ones(n_offset, 1);
x_star(n_offset+1:N+n_offset, 2) = z(1:4:N*nx);
x_star(N+n_offset+1:N+2*n_offset, 2) = x_star(n_offset+N, 2) * ones(n_offset, 1);
x_star(n_offset+1:N+n_offset, 3) = z(2:4:N*nx);
x_star(n_offset+1:N+n_offset, 4) = z(3:4:N*nx);
x_star(n_offset+1:N+n_offset, 5) = z(4:4:N*nx);
```

```
62
63 %% Plot simulated system
64 fig = figure(1); clf(1);box  on;
65 set(gca,'FontSize',11)
66 sim_travel = z(1:nx:N*nx);
67 sim_pitch  = z(3:nx:N*nx);
68 hold all;
69 plot(t, (180/pi)*x_star(:, 2), 'LineWidth', 2);
70 plot(t, (180/pi)*u_star(:, 2), 'LineWidth', 2);
71 legend('Optimal travel', 'Optimal pitch reference');
72 xlabel('Time [s]');
73 ylabel('Angle [deg]');
74 title(sprintf('Simulation of system over %d-length horizon', N));
75
76 % Plot results
77 %load('measurements.mat');
78 %save(sprintf('../../measurements/day3/measurements_q_%d_%d_%d_%d.mat',Q_LQR(1,1),Q_LQR(2,2),Q_LQ
79 load (sprintf('../../measurements/day3/measurements_q_%d_%d_%d_%d.mat',Q_LQR(1,1),Q_LQR(2,2),Q_LQ
80 t_real = measurements(1,:);
81 travel = (180/pi)*measurements(2,:);
82 pitch = (180/pi)*measurements(4,:);
83 plot(t_real,travel, 'LineWidth', 2,'LineStyle','--');
84 plot(t_real, pitch, 'LineWidth', 2, 'LineStyle', '--');
85 legend('Opt travel ref', 'Opt pitch ref','Real Travel', 'Real Pitch');
86 title('Simulated optimal trajectory with feedback', 'FontSize',14,'FontWeight','normal')
87 plot(t, zeros(2*n_offset+N,1), '--','Color', [0 0 0]);
88 line([offsetTime offsetTime],get(gca,'YLim'),'Color','Black','LineWidth',1);
89 line([offsetTime+N*h offsetTime+N*h],get(gca,'YLim'),'Color','Black','LineWidth',1);
90 xlim([0 (N+2*n_offset)*h]);
91 % Set the dimensions of the figure
92 set(fig, 'units','centimeters');
93 pos = get(gcf, 'position');
94 set(gcf, 'position', [pos(1), pos(2), 15, 9]);
```

A.3 Day 4 open-loop

```
run('../init.m');
global N alpha beta travel_t nx
% Adjustable parameters
x0 = [pi 0 0 0 0 0]';        % Initial state
h  = 0.25;                   % Discretization timestep
N  = 60;                     % Length of horizon
offsetTime = 5;           % Init time at start of simulation
n_offset = offsetTime/h; % Deadzone at start and end (timesteps)
nx = 6;                      % Number of states of system
nu = 2;                      % Number of inputs of system
nz = N*(nx+nu);              % Size of z
Q  = diag([1 0 0.1 0 0 0]); % State penalty weights
R  = diag([1 1]);            % Input penalty weight
% Non-linear elevation constraint params
alpha     = 0.2;
beta      = 20;
travel_t  = 2*pi/3;
% System state and input bounds
pitch_lim = 30*pi/180;
pitch_rate_lim = 30*pi/180;
x_max = [+inf +inf +pitch_lim +pitch_rate_lim +inf +inf]';
x_min = [-inf -inf -pitch_lim -pitch_rate_lim -inf -inf]';
u_max = [+pitch_lim +inf]';
u_min = [-pitch_lim -inf]';
% Continuous-time system matrices
Ac = [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];

Bc = [0 0 ; 0 0 ; 0 0 ; K_1*K_pp 0 ; 0 0 ; 0 K_3*K_ep];
% Discrete-time system matrices
A = eye(6) + h * Ac;
B = h * Bc;
% Solve QP
Aeq             = gena2(A, B, N, nx, nu);
Beq             = zeros(N*nx, 1);
Beq(1:nx)       = A*x0;
H               = genq2(Q, R, N, N);
[lb, ub]        = genbegr2(N, N, x_min, x_max, u_min, u_max);
lb(nx*(N-1)+1) = 0;
ub(nx*(N-1)+1) = 0;
f = @(z) z' * H * z;
final_e = alpha*exp(-beta*(pi-travel_t)^2);
final_x = [0 0 0 0 final_e+0.1 0]';
final_u = [0 final_e+0.1]';
z0 = [x0;repmat(final_x, N-1, 1);repmat(final_u, N, 1)];
options = optimset('Algorithm','sqp');
z = fmincon(f, z0, [], [], Aeq, Beq, lb, ub, @confun,options);
% Create Simulink inputs
t = (0:N+2*n_offset-1) * h;
u = [z(N*nx+1:nu:nz) z(N*nx+2:nu:nz)];
u_star = zeros(N+2*n_offset, nu+1);
u_star(:, 1) = t;
u_star(n_offset+1:N+n_offset, 2) = u(:,1);
u_star(n_offset+1:N+n_offset, 3) = u(:,2);
x_star = zeros(N+2*n_offset, nx+1);
x_star(:, 1) = t;
x_star(1:n_offset, 2) = pi * ones(n_offset, 1);
```

```
x_star(n_offset+1:N+n_offset, 2) = z(1:nx:N*nx);
x_star(N+n_offset+1:N+2*n_offset, 2) = x_star(n_offset+N, 2) * ones(n_offset, 1);
x_star(n_offset+1:N+n_offset, 3) = z(2:nx:N*nx);
x_star(n_offset+1:N+n_offset, 4) = z(3:nx:N*nx);
x_star(n_offset+1:N+n_offset, 5) = z(4:nx:N*nx);
x_star(n_offset+1:N+n_offset, 6) = z(5:nx:N*nx);
x_star(n_offset+1:N+n_offset, 7) = z(6:nx:N*nx);

%% Plot simulated trajectory and input
fig = figure(1); clf(1);
set(gca,'FontSize',11)

subplot(2,1,1);
hold all;
sim_travel = z(1:nx:N*nx);
sim_elev   = z(5:nx:N*nx);
pitch_ref  = z(N*nx+1:nu:N*(nx+nu));
elev_ref   = z(N*nx+2:nu:N*(nx+nu));
plot(t, x_star(:,2)*180/pi, 'LineWidth', 2);
plot(t, u_star(:,2)*180/pi,  'LineWidth', 2);
legend('Opt travel traj', 'Opt pitch ref');
xlabel('Time [s]');
ylabel('Angle [deg]');
subplot(2,1,2);
hold all;
c = alpha*exp(-beta*(sim_travel-travel_t).^2);
plot(t, x_star(:,6)*180/pi,   'LineWidth', 2);
plot(t, u_star(:,3)*180/pi,   'LineWidth', 2);
plot(t, [zeros(n_offset,1); c*180/pi; zeros(n_offset,1)], '--', 'LineWidth', 2);
legend('Opt elev traj', 'Opt elev ref', 'Constraint');
xlabel('Time [s]');
ylabel('Angle [deg]');

% Plot results
%load ('measurements.mat');
%save ('../../measurements/day4_openloop/measurements.mat','measurements');
load('../../measurements/day4_openloop/measurements');
t_real = measurements(1,:);
travel = (180/pi)*measurements(2,:);
pitch = (180/pi)*measurements(4,:);
elevation = (180/pi)*measurements(6,:);
subplot(2,1,1);
hold all;
plot(t_real,travel, 'LineWidth', 2,'LineStyle','--');
plot(t_real, pitch, 'LineWidth', 2, 'LineStyle', '--');
legend('Opt travel traj', 'Opt pitch ref','Real Travel', 'Real Pitch');
title('Simulated optimal trajectory without feedback','FontSize',14,'FontWeight','normal');
line([offsetTime offsetTime],get(gca,'YLim'),'Color','Black','LineWidth',1);
line([offsetTime+N*h offsetTime+N*h],get(gca,'YLim'),'Color','Black','LineWidth',1);
xlim([4 (N+2*n_offset)*h]); ylim([-140 205]);
subplot(2,1,2);
hold all;
plot(t_real,elevation, 'LineWidth', 2,'LineStyle','--');
legend('Opt elev traj', 'Opt elev ref', 'Constraint','Real elevation');
line([offsetTime offsetTime],get(gca,'YLim'),'Color','Black','LineWidth',1);
line([offsetTime+N*h offsetTime+N*h],get(gca,'YLim'),'Color','Black','LineWidth',1);
xlim([4 (N+2*n_offset)*h]); ylim([-5 14]);
% Set the dimensions of the figure
set(fig, 'units','centimeters');
pos = get(gcf, 'position');
set(gcf, 'position', [pos(1), pos(2), 15, 12]);
```

A.4 Day 4 closed-loop

```
run('../init.m');
global N alpha beta travel_t nx
% Adjustable parameters
x0 = [pi 0 0 0 0 0]';        % Initial state
h  = 0.25;                   % Discretization timestep
N  = 60;                     % Length of horizon
offsetTime = 5;           % Init time at start of simulation
n_offset = offsetTime/h; % Deadzone at start and end (timesteps)
nx = 6;                      % Number of states of system
nu = 2;                      % Number of inputs of system
nz = N*(nx+nu);              % Size of z
Q  = diag([1 0 0 0 0 0]); % State penalty weights
R  = diag([1 1]);            % Input penalty weight
% Non-linear elevation constraint params
alpha     = 0.2;
beta      = 20;
travel_t  = 2*pi/3;
% System state and input bounds
pitch_lim = 30*pi/180;
pitch_rate_lim = 30*pi/180;
x_max = [+inf +inf +pitch_lim +pitch_rate_lim +inf +inf]';
x_min = [-inf -inf -pitch_lim -pitch_rate_lim -inf -inf]';
u_max = [+pitch_lim +inf]';
u_min = [-pitch_lim -inf]';
% Continuous-time system matrices
Ac = [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];

Bc = [0 0 ; 0 0 ; 0 0 ; K_1*K_pp 0 ; 0 0 ; 0 K_3*K_ep];
% Discrete-time system matrices
A = eye(6) + h * Ac;
B = h * Bc;
% Solve QP
Aeq             = gena2(A, B, N, nx, nu);
Beq             = zeros(N*nx, 1);
Beq(1:nx)       = A*x0;
H               = genq2(Q, R, N, N);
[lb, ub]        = genbegr2(N, N, x_min, x_max, u_min, u_max);
lb(nx*(N-1)+1) = 0;
ub(nx*(N-1)+1) = 0;
f = @(z) z' * H * z;
final_e = alpha*exp(-beta*(pi-travel_t)^2);
final_x = [0 0 0 0 final_e+0.1 0]';
final_u = [0 final_e+0.1]';
z0 = [x0;repmat(final_x, N-1, 1);repmat(final_u, N, 1)];
options = optimset('Algorithm','sqp');
z = fmincon(f, z0, [], [], Aeq, Beq, lb, ub, @confun,options);
% LQR
Q_LQR = diag([20 1 1 1 30 10]);
R_LQR = diag([1 1]);
[K, S, E] = dlqr(A,B,Q_LQR,R_LQR);
% Create Simulink inputs
t = (0:N+2*n_offset-1) * h;
u = [z(N*nx+1:nu:nz) z(N*nx+2:nu:nz)];
u_star = zeros(N+2*n_offset, nu+1);
u_star(:, 1) = t;
u_star(n_offset+1:N+n_offset, 2) = u(:,1);
```

```
u_star(n_offset+1:N+n_offset, 3) = u(:,2);
x_star = zeros(N+2*n_offset, nx+1);
x_star(:, 1) = t;
x_star(1:n_offset, 2) = pi * ones(n_offset, 1);
x_star(n_offset+1:N+n_offset, 2) = z(1:nx:N*nx);
x_star(N+n_offset+1:N+2*n_offset, 2) = x_star(n_offset+N, 2) * ones(n_offset, 1);
x_star(n_offset+1:N+n_offset, 3) = z(2:nx:N*nx);
x_star(n_offset+1:N+n_offset, 4) = z(3:nx:N*nx);
x_star(n_offset+1:N+n_offset, 5) = z(4:nx:N*nx);
x_star(n_offset+1:N+n_offset, 6) = z(5:nx:N*nx);
x_star(n_offset+1:N+n_offset, 7) = z(6:nx:N*nx);

%% Plot simulated trajectory and input
fig = figure(1); clf(1);
set(gca,'FontSize',11)
subplot(2,1,1); hold all;
sim_travel = z(1:nx:N*nx);
sim_elev   = z(5:nx:N*nx);
pitch_ref  = z(N*nx+1:nu:N*(nx+nu));
elev_ref   = z(N*nx+2:nu:N*(nx+nu));
plot(t, x_star(:,2)*180/pi, 'LineWidth', 2);
plot(t, u_star(:,2)*180/pi,  'LineWidth', 2);
legend('x_t^*','u_p^*');
xlabel('Time [s]');
ylabel('Angle [deg]');
subplot(2,1,2); hold all;
c = alpha*exp(-beta*(sim_travel-travel_t).^2);
plot(t, x_star(:,6)*180/pi,   'LineWidth', 2);
plot(t, u_star(:,3)*180/pi,   'LineWidth', 2);
plot(t, [zeros(n_offset,1); c*180/pi; zeros(n_offset,1)], '--', 'LineWidth', 2);
legend('x_e^*', 'u_e^*', 'Constraint');
xlabel('Time [s]');
ylabel('Angle [deg]');
% Plot results
%load ('measurements.mat'); load ('inputs.mat');
%save (sprintf('../../measurements/day4/measurements_q_%d_%d_%d_%d_%d_%d.mat', Q_LQR(1,1), Q_LQR(2
%save (sprintf('../../measurements/day4/inputs_q_%d_%d_%d_%d_%d_%d.mat', Q_LQR(1,1), Q_LQR(2,2), Q
load(sprintf('../../measurements/day4/measurements_q_%d_%d_%d_%d_%d_%d.mat', Q_LQR(1,1), Q_LQR(2,2
load(sprintf('../../measurements/day4/inputs_q_%d_%d_%d_%d_%d_%d.mat', Q_LQR(1,1), Q_LQR(2,2), Q_L
t_real = measurements(1,:);
travel = (180/pi)*measurements(2,:);
elevation = (180/pi)*measurements(6,:);
pitchInput = (180/pi)*inputs(2,:);
elevationInput = (180/pi)*inputs(3,:);
subplot(2,1,1); hold all;
plot(t_real,travel, 'LineWidth', 2,'LineStyle','--');
plot(t_real, pitchInput, 'LineWidth', 2, 'LineStyle', '--');
legend('x_t^*', 'u_p^*','x_t','u_p');
title('Simulated optimal trajectory with feedback','FontSize',14,'FontWeight','normal');
line([offsetTime offsetTime],get(gca,'YLim'),'Color','Black','LineWidth',1);
line([offsetTime+N*h offsetTime+N*h],get(gca,'YLim'),'Color','Black','LineWidth',1);
xlim([4 (N+2*n_offset)*h]); ylim([-50 210]);
subplot(2,1,2); hold all;
plot(t_real,elevation, 'LineWidth', 2,'LineStyle','--');
legend('x_e^*','u_e^*','Constraint','x_e');
line([offsetTime offsetTime],get(gca,'YLim'),'Color','Black','LineWidth',1);
line([offsetTime+N*h offsetTime+N*h],get(gca,'YLim'),'Color','Black','LineWidth',1);
xlim([4 (N+2*n_offset)*h]); ylim([-5 14]);
% Set the dimensions of the figure
set(fig, 'units','centimeters'); pos = get(gcf, 'position'); set(gcf, 'position', [pos(1), pos(2),
```

B Simulink Diagrams

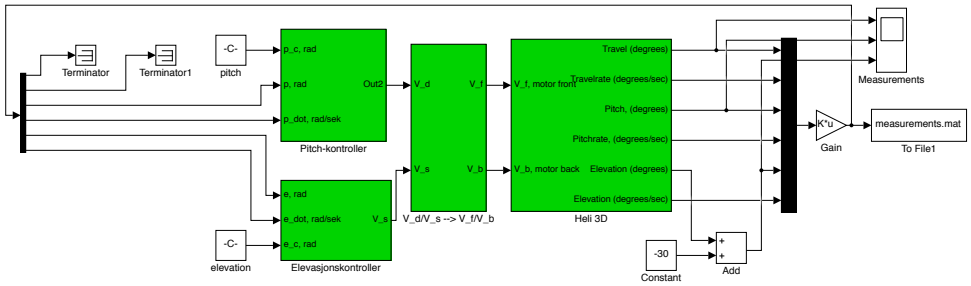

Figure 9: Simulink optimal open-loop model

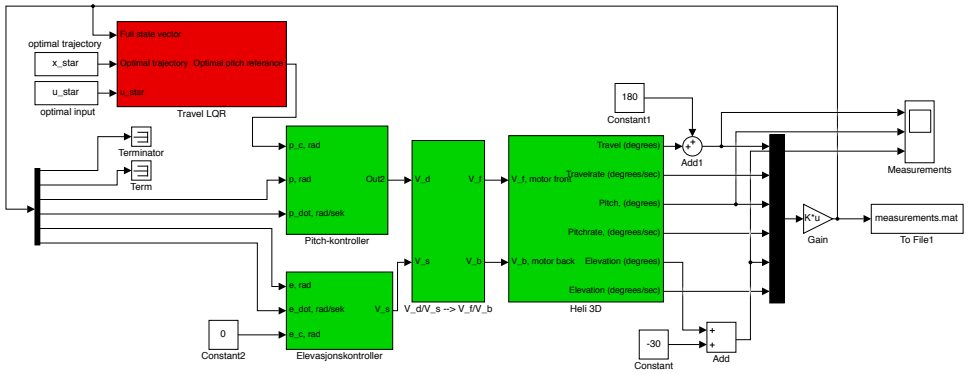

Figure 10: Simulink optimal closed-loop model

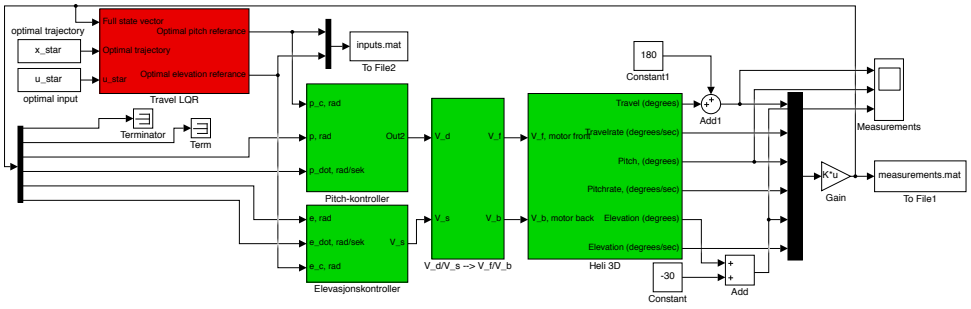

Figure 11: Simulink optimal closed-loop model in two dimensions

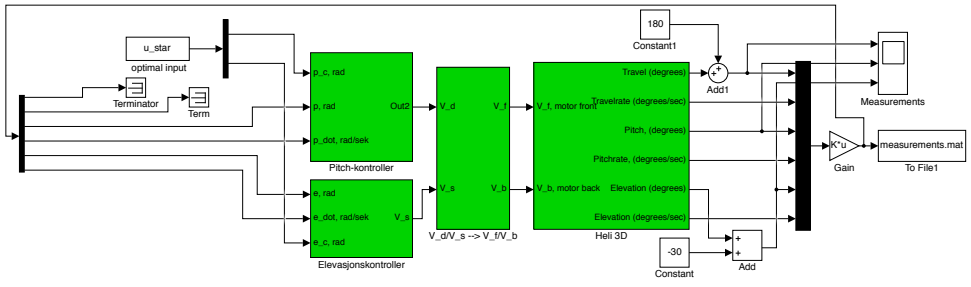

Figure 12: Simulink optimal open-loop model in two dimensions

Bibliography

Jesmani, M. (2015). Ttk4135 optimization and control, helicopter lab.