TTK 4135 Optimization and Control Helicopter Lab

Henrik Enger Reimers 552364
Erik Fougner Arnesen 598690

April, 2024

NTNU

Department of Engineering Cybernetics

Contents

1 Introduction

1.1 Objective

The objective of this lab is to utilize optimization techniques to control a system. The assignment specifies that a study on a 3- DOF helicopter is to be conducted. Given a mathematical model of the system, the goal is to study control behavior with different controllers. First we implement optimal control of pitch and travel, both with and without feedback. This was achieved by minimizing a quadratic optimization problem. The feedback was applied using a linear quadratic controller. In the last part of the assignment optimal control of pitch, travel and elevation with feedback was performed. A nonlinear inequality constraint was applied on the elevation during this task. $MATLAB$ was utilized to calculate the optimal control parameters and $SIMULINK$ was used to control the helicopter.

1.2 System description

The physical helicopter system consists of a base with an arm attached. The arm has a counterweight in one end and two rotors in the other end. The arm can rotate around the base (travel) and move up and down (elevation). A difference in voltage given to the rotors can be used to control rotation around an axis normal to the arm (pitch). These angles are measured using encoders. A cross- section of the helicopter can be seen in figure 1. The parameters and variables for the model can be seen in table 1 and 2.

Table 1: Parameters and values

Symbol	Parameter	Value	Unit
l_a	Distance from elevation axis to helicopter body	0.63	m
l_h	Distance from pitch axis to motor	0.18	m
K_f	Force constant motor	0.25	N/V
J_e	Moment of inertia for elevation	0.83	kg mˆ2
J_t	Moment of inertia for travel	0.83	kg mˆ2
J_p	Moment of inertia for pitch	0.034	kg mˆ2
mh	Mass of helicopter	1.05	kg
mw	Balance weight	1.87	kg
mg	Effective mass of the helicopter	0.05	kg
K_p	Force to lift the helicopter from the ground	0.49	N

Table 2: Variables

Symbol	Variable
p	Pitch
p_c	Setpoint for pitch
λ	Travel
r	Speed of travel
r_c	Setpoint for speed of travel
e	Elevation
e_c	Setpoint for elevation
V_f	Voltage, motor in front
V_b	Voltage, motor in back
V_d	Voltage difference, Vf - Vb
V_s	Voltage sum, Vf + Vb
$K_{pp}, K_{pd}, K_{ep}, K_{ed}$	Controller gains
T_g	Moment needed to keep the helicopter flying

1.3 Derivation of mathematical model

The equations for the mathematical model of the helicopter were derived using Newtons second law for rotation and moment balances. In combination with the sketch in figure 2 it is possible to derive a model of the elevation, pitch and travel.[1]

1.3.1 Elevation model

$$J\ddot{e} = l_a K_f V_s - T_g \tag{1a}$$

where

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

The following PD- controller is used to control the elevation:

$$V_s = K_{ep}(e_c - e) - K_{ed}\dot{e}, \quad K_{ep}, K_{ed} > 0 \tag{1}$$

This results in the following system:

$$\ddot{e} = -K_3 K_{ed}\dot{e} + K_3 K_{ep} e - \frac{T_g}{J_e} + K_3 K_{ep} e_c \tag{2}$$

Finally an integral term is added resulting in an PID- controller. It is assumed that the integral term nullify the effect from the constant term -$\frac{T_g}{J_e}$. The model for the elevation is then:

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

1.3.2 Pitch model

$$J_p \ddot{p} = K_f l_h V_d \tag{4}$$

where

$$\ddot{p} = K_1 V_d \quad K_1 = \frac{K_f l_h}{J_p} \tag{5}$$

The following PD - controller is used to control the pitch angle:

$$V_d = K_{pp}(p_c - p) - K_{pd}\dot{p}, \quad K_{pp}, K_{pd} > 0 \tag{6}$$

The model for the pitch then results in:

$$\ddot{p} + K_1 K_{pd}\dot{p} + K_1 K_{pp} p = K_1 K_{pp} p_c \tag{7}$$

1.3.3 Travel model

$$J_t \ddot{r} = -K_p l_a \sin p \tag{8}$$

It is assumed that angle p is small, resulting in $\sin p = p$. Then,

$$\dot{r} = K_2 p, \quad K_2 = \frac{K_p l_a}{J_t} \tag{9}$$

1.3.4 Summary of relevant equations

$$\ddot{e} = -K_3 K_{ed}\dot{e} + K_3 K_{ep} e - \frac{T_g}{J_e} + K_3 K_{ep} e_c \tag{10}$$

$$\ddot{p} + K_1 K_{pd}\dot{p} + K_1 K_{pp} p = K_1 K_{pp} p_c \tag{11}$$

$$\dot{\lambda} = r \tag{12}$$

$$\dot{r} = -K_2 p \tag{13}$$

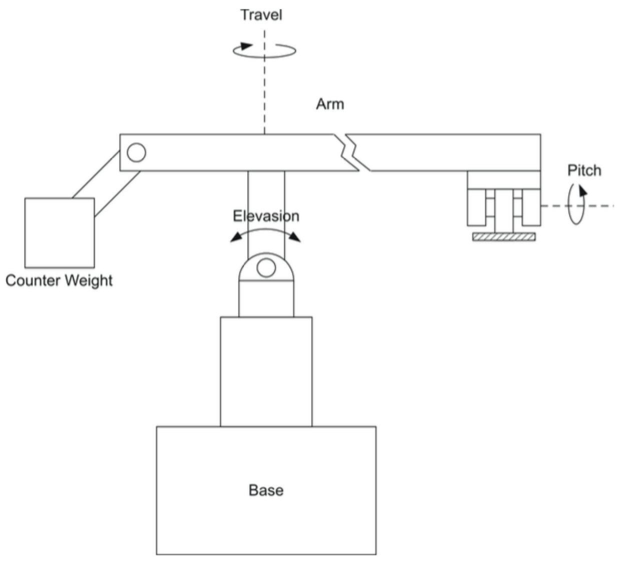


Figure 1: Cross-section of the helicopter. [1]

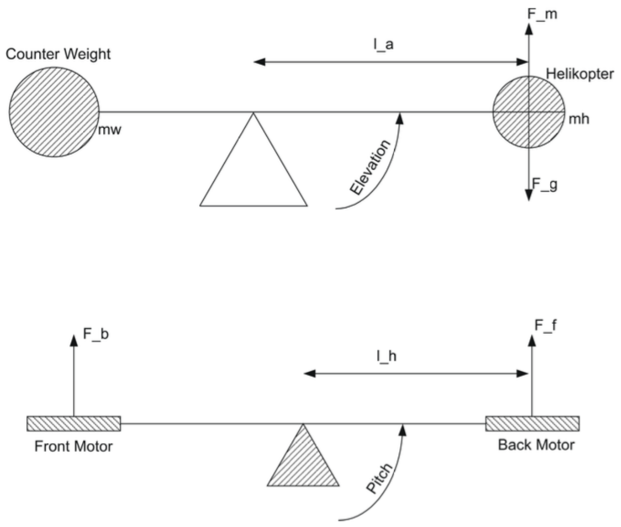


Figure 2: Sketch of the helicopter. [1]

2 Optimal Control of Pitch/Travel without Feedback

In the first part of the lab assignment the task is to calculate an optimal trajectory $x*$ and input sequence $u*$. This is achieved by minimizing the given quadratic programming problem. The optimal input sequence was used to drive the helicopter from an initial starting position x_0 to a predefined reference point x_f. It is assumed that elevation is equal to zero.

2.1 The continuous model

The mathematical model derived in section 1.3 can be expressed in continuous state-space form:

$$\begin{aligned} \dot{x} &= A_c x + B_c u \\ x &= [\lambda \quad r \quad p \quad \dot{p}]^T \\ u &= p_c \end{aligned} \tag{14}$$

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1K_{pp} \end{bmatrix} p_c \tag{15}$$

2.2 The discretized model

In order to discretize the model, the forward Euler method is used. The method in state- space form is defined as the following:

$$x_{k+1} = \underbrace{(I - \Delta t A_c)}_{A_d} x_k + \underbrace{\Delta t B_c u_k}_{B_d}, \quad \Delta t = 0.25s \tag{16}$$

where

$$A_d = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \Delta t & 1 & 0 & 0 \\ 0 & -\Delta t K_2 & 1 & -\Delta t K_1 K_{pp} \\ 0 & 0 & \Delta t & 1 - \Delta t K_1 K_{pd} \end{bmatrix}, \quad B_d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \Delta t K_1 K_{pp} \end{bmatrix} \tag{17}$$

2.3 Calculating the optimal path with quadratic programming

To obtain the optimal trajectory the following objective function needs to be minimized:

$$\Phi = \sum_{i=1}^{N} (\lambda_i - \lambda_f)^2 + q p_{ci}^2, \quad q \geq 0 \tag{18}$$

Using equation 33 it is possible to derive the Hessian matrix (Q):

$$Q = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{19}$$

The optimal path is to be calculated from the initial starting point of $x_o = \begin{bmatrix} \pi & 0 & 0 & 0 \end{bmatrix}^T$ to the reference point $x_f = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T$. To achieve this the following inequality constrained quadratic problem (QP) needs to be solved:

$$\min_{z \in R^n} \quad f(z) = \frac{1}{2} z^T G z, \quad s.t \quad A_{eq} z = b_{eq}, \tag{20}$$

G, A_{eq} and b_{eq} is defined as:

$$G = \begin{bmatrix} Q & 0 & \dots & \dots & \dots & 0 \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & Q & \ddots & & \vdots \\ \vdots & & \ddots & q & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & \dots & 0 & q \end{bmatrix} \tag{21}$$

$$A_{eq} = \begin{bmatrix} I & 0 & \dots & \dots & 0 & -B & 0 & \dots & \dots & 0 \\ -A & I & \ddots & & \vdots & 0 & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 & \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & 0 & -A & I & 0 & \dots & \dots & 0 & -B \end{bmatrix}, \quad b_{eq} = \begin{bmatrix} Ax_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{22}$$

and inequality constraint on the pitch angle:

$$|p_k| \leq \frac{\pi}{6}, \quad k \in \{1, ..., N\} \tag{23}$$

The optimization problem was solved using the quadprog function in matlab. First we had to load a initialization file that were handed out to us. This file contained all the necessary variables that were required for the model definition, like K_{pp} and K_1. Next, we discretize the model as described in 2.2.

The next step is then to declare x0, N and a z vector containing all the variables that we want to optimize:

$$z = (x_1^T ..., x_N^T, u_1^T ..., u_{N-1}^T), \quad N = 100 \tag{24}$$

Several helper function was given to us to help with generation of various matrices. Firstly we utilized gen_constraints to produce constraints for the measurements and inputs. To fill in the Q and A_eq matrices the helper functions gen_q and gen_aeq was used.

Lastly we called the quadprog function wich solved the QP- problem and gave us the optimal input for the helicopter. The input values was then extracted and imported into our *SIMULINK* model, so we could compare the simulation in *MATLAB* with the physical system. Both the *MATLAB* code and the *SIMULINK* model can be found in the appendix

2.4 Results

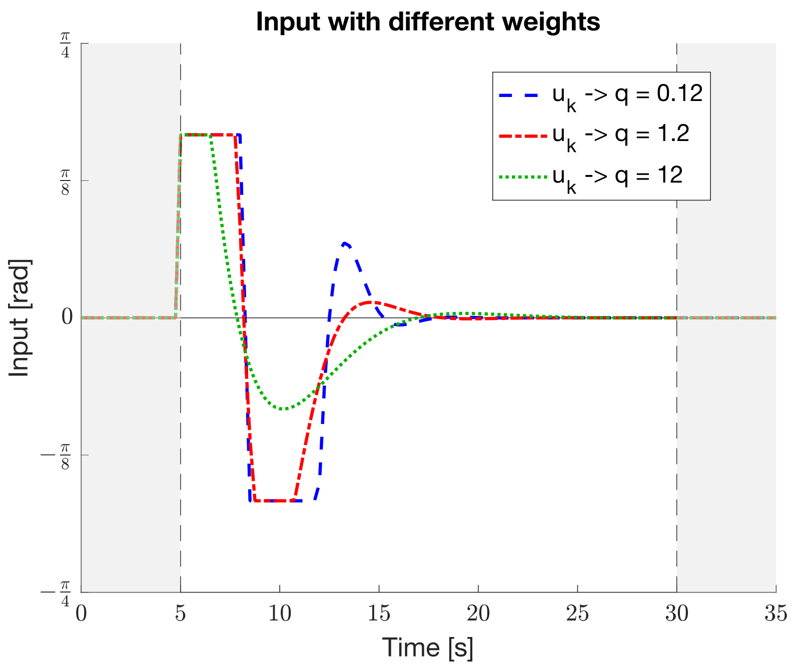


Figure 3: Optimized input for different weights.

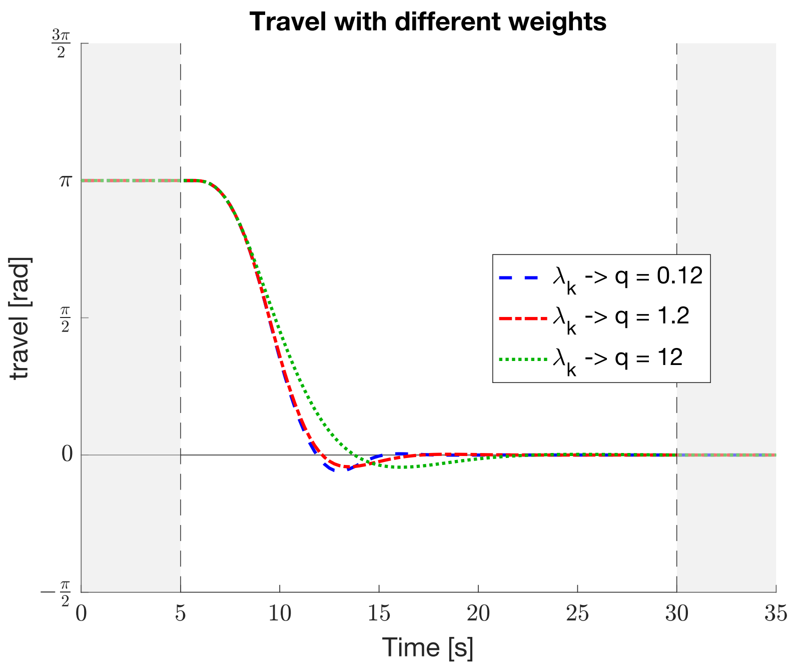


Figure 4: Plot of travel for different weights.

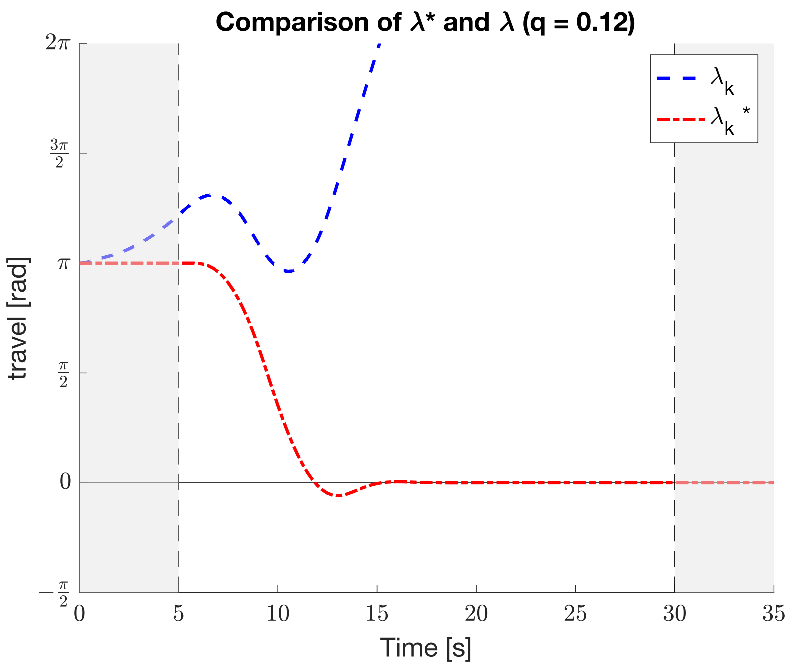


Figure 5: Optimal travel vs. recorded travel.

Increasing the weight q gave a smoother input sequence, and this lead to a more gentle control of the helicopter. It is possible to observe in figure 3 that $q = 12$ converges to 0 slowly. Decreasing q results in aggressive control, with sudden changes in the input and faster convergence to 0.

Looking at figure 5 it is possible to see that the helicopter never reaches the intended reference point x_f. This is because the controller in use doesn't rely on feedback, therefore it assumes stationary when it reaches the reference point. In practice the helicopter still has momentum and continues past x_f. Implementing a closed- loop solution can help remedy this overshoot.

3 Optimal Control of Pitch/Travel with Feedback (LQ)

3.1 LQ controller

An LQ controller is short for Linear-Quadratic controller, this is a type of optimal controller used to design control systems. It's goal is to minimize the quadratic criteria for a linear model [1]. Using the previously calculated optimal trajectory from exercise 2, we have the optimal input sequence u* and the optimal trajectory x*. Utilizing these variables we can integrate feedback into the control system

$$u_k = u_k^* - K(x_k - x_k^*) \tag{25}$$

Here the input sequence u* is calculated using the systems ability to follow then calculated optimal trajectory. Whenever a deviation is observed form the optimal trajectory x*, this will then effect the variable feedback back to the system. The final unknown variable K is the optimal gain matrix, which can be calculated using the MATLAB function *dlqr* that minimize the quadratic objective function.

$$J = \sum_{i=0}^{\infty} \Delta x_{i+1}^T Q \Delta x_{i+1} + \Delta u_i^T R \Delta u_i, \quad Q \geq 0, R \geq 0 \tag{26}$$

for the linear model

$$\Delta x_{i+1} = A\Delta x_1 + B\Delta u_i \tag{27}$$

not including the inequality constraints. Here, Δx and Δu are deviations from the optimal trajectory

$$\Delta x = x - x* \tag{28}$$

$$\Delta u = u - u* \tag{29}$$

Here the **Q** and **R** are the weighted matrices. The **Q** matrix weighs the importance to minimize the deviations of the system's states from the optimal trajectory, while the **R** matrix weighs the importance to minimize the control effort.

$$Q = \begin{bmatrix} q_1 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 \\ 0 & 0 & q_3 & 0 \\ 0 & 0 & 0 & q_4 \end{bmatrix}, \quad R = r_1$$

When placing a large value in a specific position in the **Q** matrix such as q_1, we place a larger importance on the state travel. This will then penalize any deviation from the corresponding optimal state more heavily, making the controller working harder to keep that state closer to the desired value. lower values on the other hand will lead to more freedom to deviation to the state. But in the **R** matrix case, we place importance to minimize the control effort. If we place a high value on r_1 we penalize more for using more control, making the controller more conservative on using control input.

In our case we want the controller to penalize deviation of travel and travel rate, while allowing states such as pitch and pitch rate to be more free. Our goal is to change the travel state to a specific value, therefore it is beneficial to be strict, ensuring a successful trajectory and stable final position. For **R** on the other hand we may want to have a lower value on the state to making the controller less conservative. After multiple tests we ended up with the **Q** and **R** matrix.

$$Q = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix}, \quad R = 0.1$$

The resulting optimal gain matrix K becoming

$$K = [-3.9054, -8.5146, 3.4289, 0.8503]$$

3.2 Model Predictive Control

Implementing a MPC, where the idea is to constantly updating the model predictions using every new measurment. To implement the MPC it would entail having to sole the QP problem at every timestep. Some of the advantages to MPC is that it takes into consideration constraint, such as physical limitation due to limits to actuator. As well as being intuitive and simple control policy as noted in [2]. Some Advantages that are also notable is MPC outperforming LQ in trajectory tracking, while comparably having a reduced actuator wear. This is due to MPC being characterized having smoother changes in control signals, noted in [3]. But there are some disadvantages to Model predictive control(MPC) as well. It will require having to solve the optimization problem at every time step. leading to needing larger computational resources by having a higher computational load, as noted in[2]. This is leading to MPC for fast real-time applications being limited, when compared to LQ due to only having solve the optimization problem once. Taking into account how fast the helicopter moves, and how fast the sensor readings that are being sent to the controller. In summation, MPC offers greater flexibility particularly in constrained, and multivariable systems, at the cost of higher computational requirements. LQR provides a simpler, computationally efficient for without taking into account constraints, making the choice between MPC and LQR dependent on the specific requirements of the control problem. implementing the MPC controler would potentially look something like in figure 6

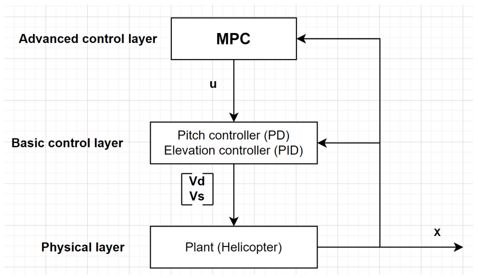


Figure 6: Diagram of MPC implementation

3.3 Results

When comparing the results from different experiments, we see that our LQR controller preforms very well in following the optimal trajectory in travel. When comparing to pitch it travels well in the start but suffers from steady state error at the end. The fact that the LQ controller with feedback tracks better then the controller without feedback should also come as a surprise.

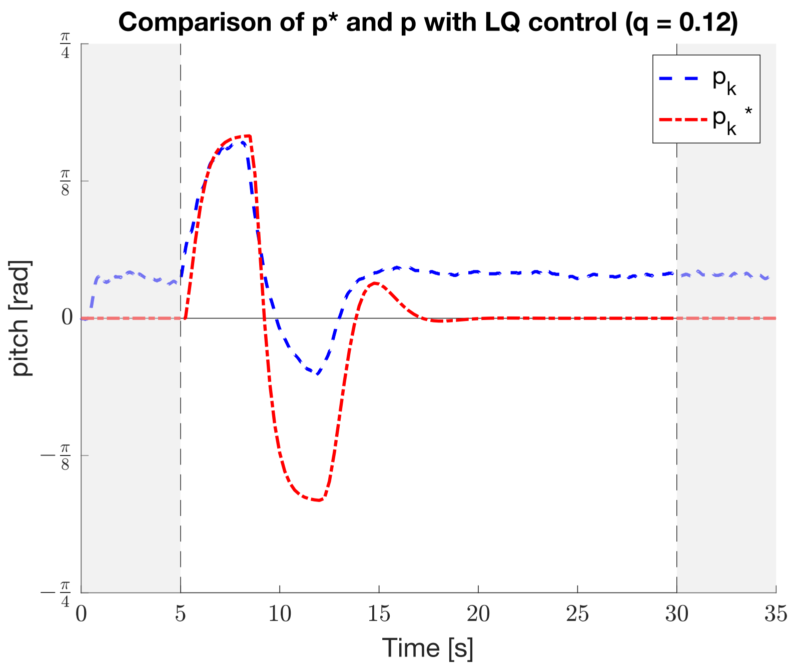


Figure 7: controller following optimal trajectory in pitch.

3.4 MATLAB and Simulink

Listing 1: MATLAB code for calculating the optimal gain matrix K

```
%% Q, R and K matrix
q_1 = 5;
q_2 = 1;
q_3 = 0.1;
q_4 = 0.1;
Q_matrix = diag([q_1,q_2,q_3,q_4]);

R = 0.1;
K = dlqr(A1,B1,Q_matrix,R);
```

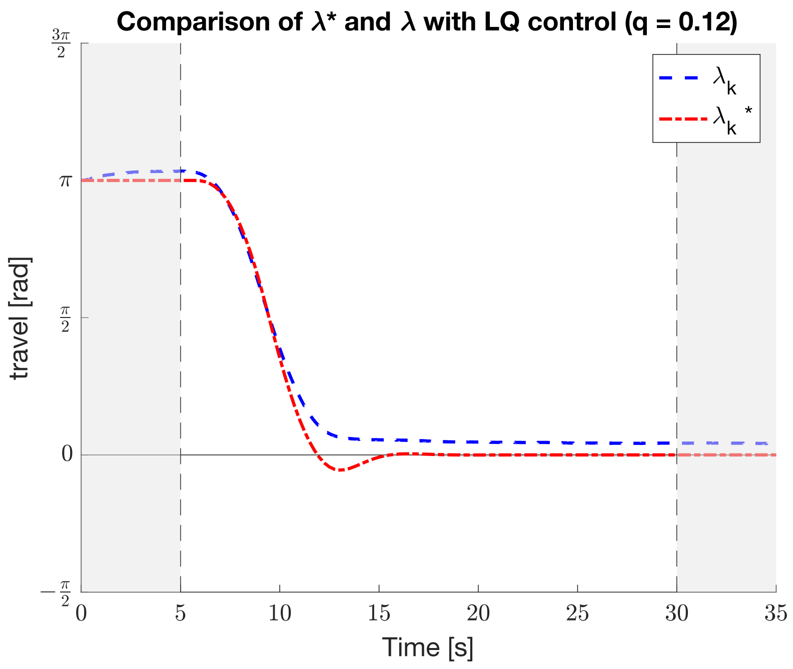


Figure 8: controller following optimal trajectory in travel.

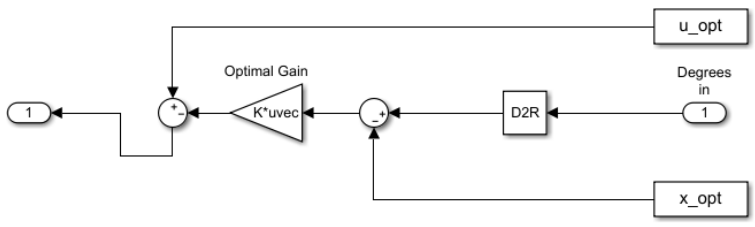


Figure 9: Implementation of LQ controller.

4 Optimal Control of Pitch/Travel and Elevation with Feedback

In the last section of this report the dynamics of elevation e is included. The goal is now to calculate an optimal path from x_0 to x_f past an obstacle which require a change in the elevation angle. This obstacle is implemented in the problem via a nonlinear constraint. This requires the use of a nonlinear solver to find the optimal path of the two dimensions.

4.1 Expansion of the continuous model

In order to accommodate the new conditions, two new states e and $\dot{e}$ needs to be incorporated. A new input controlling the elevation is also included. The continuous state-space form is now expressed as the following:

$$\begin{aligned} \dot{x} &= A_c x + B_c u \\ x &= [\lambda \quad r \quad p \quad \dot{p} \quad e \quad \dot{e}]^T \\ u &= [p_c \quad e_c]^T \end{aligned} \tag{30}$$

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3K_{ep} & -K_3K_{ed} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3K_{ep} \end{bmatrix} \begin{bmatrix} p_c \\ e_c \end{bmatrix} \tag{31}$$

4.2 The discretized model

In order to discretize the model, the forward Euler method described in equation 16 is used.
where

$$A_d = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & -\Delta t K_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & -\Delta t K_1 K_{pp} & 1 - \Delta t K_1 K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & -\Delta t K_e K_{ep} & 1 - \Delta t K_3 K_{ed} \end{bmatrix}, \quad B_d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \Delta t K_1 K_{pp} & 0 \\ 0 & 0 \\ 0 & \Delta t K_3 K_{ep} \end{bmatrix} \tag{32}$$

4.3 Calculating the optimal path with a nonlinear constraint on elevation

Estimating the optimal path from $x_o = \begin{bmatrix} \pi & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$ to $x_f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$ now requires a minimization of the following objective function:

$$\phi = \sum_{i=1}^{N-1} (\lambda_{i+1} - \lambda_f)^2 + q_1 p_{ci}^2 + q_2 e_{ci}^2, \quad q_1, q_2 \geq 0 \tag{33}$$

$$Q = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{34}$$

$$R = \begin{bmatrix} q_1 & 0 \\ 0 & q_2 \end{bmatrix} \tag{35}$$

a nonlinear constraint is also imposed on the elevation for every point in time:

$$c(x_k) = \alpha \exp(-(\lambda_k - \lambda_t)^2) - e_k \leq 0 \tag{36}$$

where $\alpha = 0.2$, $\beta = 20$, $\lambda_t = \frac{2\pi}{3}$

Listing 2: MATLAB code for calculating the nonlinear constraint for every timestep

```matlab
function [c, ceq] = nl_constraints(z)
    alpha = 0.2;
    beta = 20;
    lambda_t = (2*pi)/3;
    nx = 6;
    N = 40;
    c = zeros(N,1);
    for k = 1:N
        c(k) = alpha*exp(-beta*(z(1+(k - 1)*nx)-lambda_t)^2)
           - z(5+(k-1)*nx);
    end
    ceq = [];
end
```

In the previous tasks the optimization problem was solved using quadtratic programming, but in this case the problem includes a nonlinear constraint. Therefore we need to use the fmincon function to find the optimal path.

Listing 3: MATLAB code for solving the optimization problem using fmincon

```matlab
%% Solve QP problem with fmincon

f = @(z) 1/2*z'*Q*z;
opt = optimoptions('fmincon','Algorithm','sqp','MaxFunEvals',
    4000);
tic

[z, zval, exitflag] = fmincon(f,z0,[], [], Aeq, beq, vlb, vub
    , @nl_constraints, opt);
toc
```

4.4 Results

Our weighting matrix focuses on penalizing travel and elevation. There is a higher emphasis on penalizing elevation as the main objective is to clear the obstacle. We also implemented a penalizing term on pitch rate as we observed that the helicopter changed the pitch to fast, causing the helicopter to crash in the ground. After a lot tests we ended up with the following Q and R matrix:

$$Q = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{bmatrix}, \quad R = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{37}$$

The resulting optimal gain matrix is defined as:

$$K = \begin{bmatrix} -1.1643 & -3.4772 & 1.6596 & 0.6706 & \approx 0 & \approx 0 \\ \approx 0 & \approx 0 & \approx 0 & \approx 0 & 9.3768 & 8.0317 \end{bmatrix} \tag{38}$$

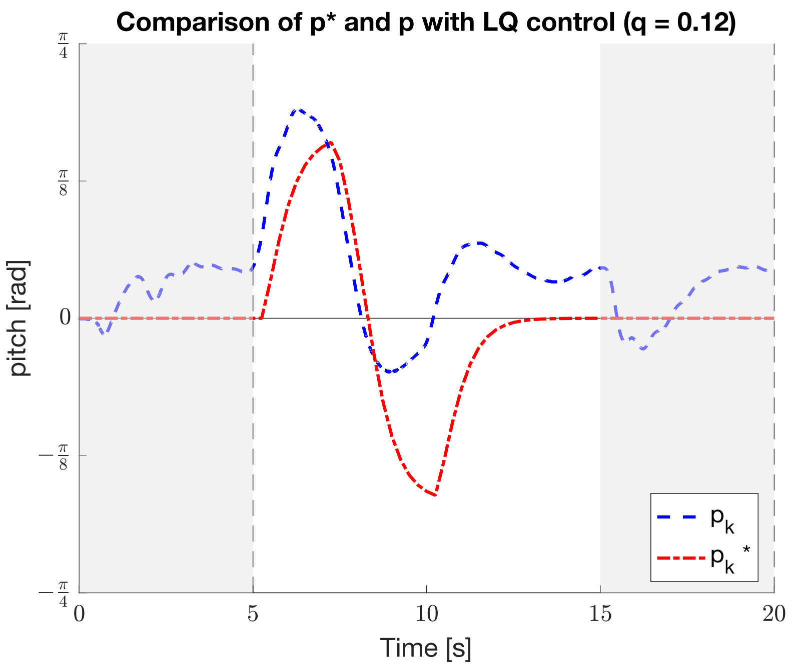


Figure 10: Optimal pitch vs. recorded pitch.

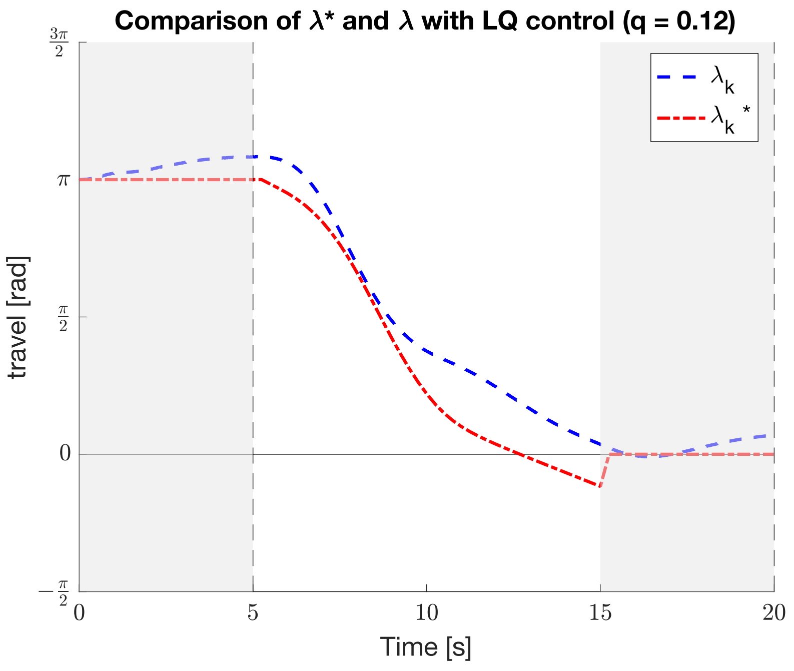


Figure 11: Optimal travel vs. recorded travel.

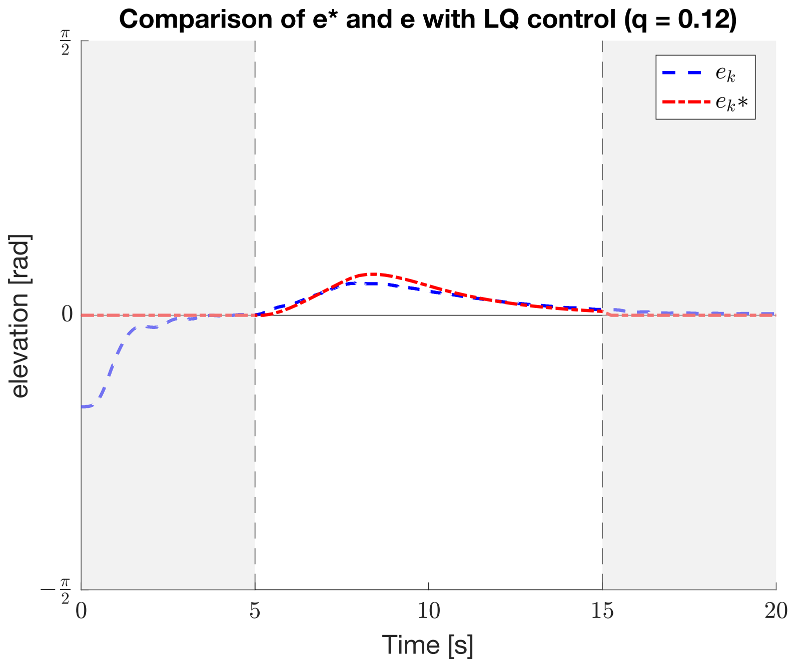


Figure 12: Optimal elevation vs. recorded elevation.

The weighting matrix resulted in an almost optimal elevation trajectory while having a less accurate trajectory for pitch and travel. This was expected as our aim was primarily to clear the simulated obstacle.

The expanded LQ- controller can be seen in the figure below:

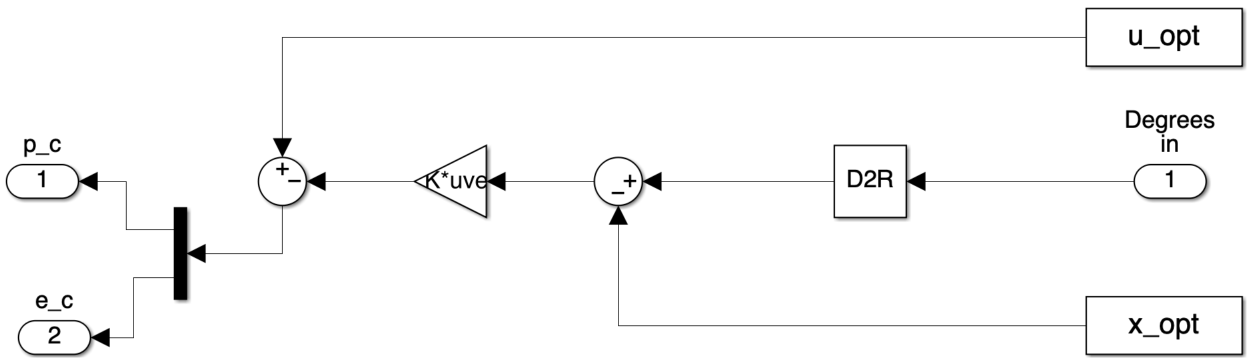


Figure 13: Final implementation of LQ controller in $SIMULINK$.

4.5 Decoupled model

In section 1.3 equation 1a states that the elevation is independent of pitch. During previous tests with the physical helicopter it is obvious that there exists some connection between these two states. This can be observed when p is set to $\frac{\pi}{2}$. Even tough the pitch is changed significantly, the elevation remains fairly stable. The model doesn't declare a relationship between pitch and elevation dynamics, but there are good two good reasons why it still works.

The first reason describes the robustness of LQ- controllers. Many control algorithms, including LQ- controllers, are inherently robust to small modeling inaccuracies or simplifications. They can still perform adequately even if the underlying model does not fully capture all the dynamics or interactions in the system. [4] In this case, the controller are able to stabilize the helicopter despite not specifically accounting for the relationship between pitch and elevation.

The second reason describes how feedback is able to to contribute to stable LQ- control. The controller relies on feedback from the actual system to adjust and stabilize the helicopter's dynamics.

Even if the model does not explicitly include the connection between pitch and elevation, the controller can learn and adapt to these interactions through feedback during operation.

4.6 Accounting for the connection between pitch and elevation

4.6.1 Rewriting the equation and state space representation

This could be a good suggestion. By explicitly incorporating the sinusoidal relationship between pitch and elevation dynamics into the model, the controller can better capture the system's behavior and potentially improve performance.

4.6.2 Adding additional terms to the objective function

This could be a bad suggestion. While incorporating more terms into the objective function might seem like a way to account for the pitch-elevation connection, it could also introduce unnecessary complexity and potentially destabilize the controller.

4.6.3 Adjusting Q-weight in the LQ controller based on pitch angle

This could be a viable suggestion. Adjusting the weighting matrix Q in the LQR controller based on the pitch angle allows the controller to prioritize certain states or dynamics during different phases of flight. It could potentially improve performance by providing better control over critical states.

A MATLAB

A.1 Initalization code

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

% Updated spring 2018, Andreas L. Flaaten
% Updated Spring 2019, Joakim R. Andersen

clear all;
close all;
clc;

% The encoder for travel for Helicopter 2 is different from
    the rest.
travel_gain = 1; %
elevation_gain = 1; %

%% Physical constants
m_h = 0.4; % Total mass of the motors.
m_g = 0.03; % Effective mass of the helicopter.
l_a = 0.65; % Distance from elevation axis to helicopter body
l_h = 0.17; % Distance from pitch axis to motor

% Moments of inertia
J_e = 2 * m_h * l_a *l_a;          % Moment of interia for
    elevation
J_p = 2 * ( m_h/2 * l_h * l_h);    % Moment of interia for
    pitch
J_t = 2 * m_h * l_a *l_a;          % Moment of interia for
    travel

% Identified voltage sum and difference
V_s_eq = 7.45;%6.3; % Identified equilibrium voltage sum.
V_d_eq = 0.2;%0.35; % Identified equilibrium voltage
    difference.

% Model parameters
K_p = m_g*9.81; % Force to lift the helicopter from the
    ground.
K_f = K_p/V_s_eq; % Force motor constant.
K_1 = l_h*K_f/J_p;
K_2 = K_p*l_a/J_t;
K_3 = K_f*l_a/J_e;
K_4 = K_p*l_a/J_e;

%% Pitch closed loop syntesis
% Controller parameters
w_p = 1.8; % Pitch controller bandwidth.
d_p = 1.0; % Pitch controller rel. damping.
K_pp = w_p^2/K_1;
K_pd = 2*d_p*sqrt(K_pp/K_1);
Vd_ff = V_d_eq;
```

```matlab

% Closed loop transfer functions
Vd_max = 10 - V_s_eq; % Maximum voltage difference
deg2rad = @(x) x*pi/180;
Rp_max = deg2rad(15); % Maximum reference step
s = tf('s');
G_p = K_1/(s^2);
C_p = K_pp + K_pd*s/(1+0.1*w_p*s);
L_p = G_p*C_p;
S_p = (1 + L_p)^(-1);

plot_pitch_response = 0;
if plot_pitch_response
    figure()
    step(S_p*Rp_max); hold on;
    step(C_p*S_p*Rp_max/Vd_max);
    legend('norm error', 'norm input')
    title('Pitch closed loop response')
end

%% Elevation closed loop analysis
% Controller parameters
w_e = 0.5; % Elevation controller bandwidth.
d_e = 1.0; % Elevation controller rel. damping.
K_ep = w_e^2/K_3;
K_ed = 2*d_e*sqrt(K_ep/K_3);
K_ei = K_ep*0.1;
Vs_ff = V_s_eq;

% Closed loop transfer functions
Vs_max = 10 - V_s_eq; % Maximum voltage sum
Re_max = deg2rad(10); % Maximum elevation step
G_e = K_3/(s^2);
C_e = K_ep + K_ed*s/(1+0.1*w_e*s) + K_ei/s;
L_e = G_e*C_e;
S_e = (1 + L_e)^(-1);

plot_elev_response = 0;
if plot_elev_response
    figure()
    step(S_e*Re_max);
    hold on;
    step(C_e*S_e*Re_max/Vs_max);
    legend('norm error', 'norm input')
    title('Elevation closed loop response')
end
```

A.2 Optimal Control of Pitch/Travel without Feedback code

```
% TTK4135 - Helicopter lab
% Hints/template for problem 2.
% Updated spring 2018, Andreas L. Flaaten

%% Initialization and model definition
init04;

delta_t = 0.25;
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]';

A_d = eye(4) + delta_t * Ac;
B_d = delta_t * Bc;

% Number of states and inputs
mx = size(A_d,2);
mu = size(B_d,2);

% 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 vector
z0 = z;                                 % Initial value for optimization

% Bounds
ul          = -1/6*pi;                       % Lower bound on control
uu          = 1/6*pi;                        % 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;                     % Setting the last input to zero
vub(N*mx+M*mu)  = 0;                     % Setting the last input to 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 = 0.12;                           % Weight on input
Q = gen_q(Q1,P1,N,M);
c = zeros(1,5*N);

%% Generate system matrices for linear model
Aeq = gen_aeq(A_d, B_d, N, mx, mu);
beq = zeros(4*N,1);
beq(1:4,1) = A_d*x0;

%% Solve QP problem with linear model
tic

[z, lambda] = quadprog(Q, c, [], [], Aeq, beq, vlb, vub, x0);
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];

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

figure(2)
subplot(511)
stairs(t,u),grid
ylabel('u')
subplot(512)
plot(t,x1,'m',t,x1,'mo'),grid
```

```
ylabel('lambda')
subplot(513)
plot(t,x2,'m',t,x2','mo'),grid
ylabel('r')
subplot(514)
plot(t,x3,'m',t,x3,'mo'),grid
ylabel('p')
subplot(515)
plot(t,x4,'m',t,x4','mo'),grid
xlabel('tid (s)'),ylabel('pdot')
```

A.3 Optimal Control of Pitch/Travel with LQ control

```
% TTK4135 - Helicopter lab
% Hints/template for problem 2.
% Updated spring 2018, Andreas L. Flaaten

%% Initialization and model definition
init; % Change this to the init file corresponding to your helicopter

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

A1 = eye(4)+ delta_t * Ac;
B1 = delta_t * Bc;

% 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          = -1/6*pi;                        % Lower bound on control
uu          = 1/6*pi;                        % 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); % hint:
   gen_constraints
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 (objective function weights in
   the QP problem)
```

```
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 = 0.12;                           % Weight on input
Q = gen_q(Q1,P1,N,M);                                  % Generate Q, hint: gen_q
c = zeros(1,5*N);                                  % Generate c, this is the 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 = zeros(4*N,1); % Generate b
beq(1:4,1) = A1*x0;

%% Solve QP problem with linear model
tic

[z, lambda] = quadprog(Q, c, [], [], Aeq, beq, vlb, vub, x0); % hint: quadprog. Type 'doc quadprog' for more info
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];

%% Q, R and K matrix
q_1 = 5;
q_2 = 1;
```

```
q_3 = 0.1;
q_4 = 0.1;
Q_matrix = diag([q_1,q_2,q_3,q_4]);

R = 0.1;
K = dlqr(A1,B1,Q_matrix,R);

%% Plotting
t = 0:delta_t:delta_t*(length(u)-1);
u_opt = timeseries(u,t);
x = [x1, x2, x3, x4];
x_opt = timeseries(x,t);

figure(2)
subplot(511)
stairs(t,u),grid
ylabel('u')

subplot(512)
plot(t,x1,'m',t,x1,'mo'),grid, hold on;
%plot(t_test,lambda_test), grid
ylabel('lambda')

subplot(513)
plot(t,x2,'m',t,x2','mo'),grid, hold on;
%plot(t_test,r_test), grid
ylabel('r')

subplot(514)
plot(t,x3,'m',t,x3,'mo'),grid, hold on;
%plot(t_test, p_test), grid
ylabel('p')

subplot(515)
plot(t,x4,'m',t,x4','mo'),grid, hold on;
%plot(t_test, pdot_test), grid
xlabel('tid (s)'),ylabel('pdot')
```

A.4 Optimal Control of Pitch/Travel and Elevation with nonlinear constraint

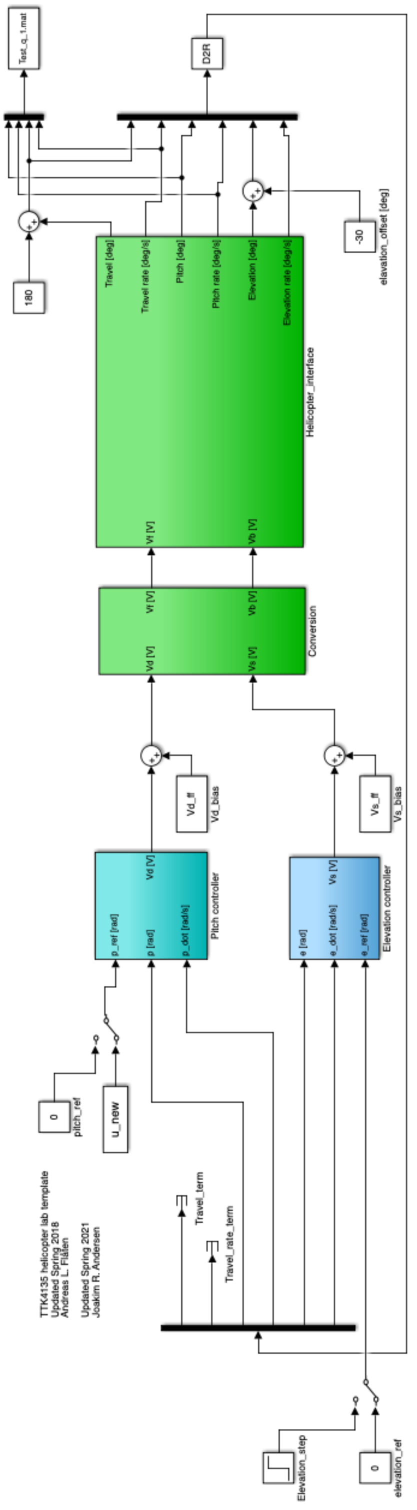
```
% TTK4135 - Helicopter lab
% Updated spring 2018, Andreas L. Flaaten

%% Initialization and model definition
init; % Change this to the init file corresponding to your helicopter

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

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];

A1 = eye(6)+ delta_t * Ac;
B1 = delta_t * Bc;

% Number of states and inputs
nx = size(A1,2);
mu = size(B1,2);
N  = 40;                                % Time horizon for states
M  = N;                                 % Time horizon for inputs
n = N*nx+N*mu;
z  = zeros(N*nx+M*mu,1);                % Initialize z for the whole
   horizon
z0 = z;                                 % Initial value for optimization

%% SQP parameters
q1 = 0.1;
q2 = q1;
alpha = 0.2;
beta = 20;
lambda_t = 2*pi/3;

%% 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
%% Bounds
ul          = -1/6*pi;                  % Lower bound on control
uu          = 1/6*pi;                   % Upper bound on control

xl      = -Inf*ones(nx,1);              % Lower bound on states (no bound)
xu      = Inf*ones(nx,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*nx+M*mu)  = 0;
vub(N*nx+M*mu)  = 0;

%% Generate the matrix Q and the vector c (objective function weights in
   the QP problem)
Q1 = zeros(nx,nx);
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
Q1(5,5) = 0;
Q1(6,6) = 0;
P = zeros(mu,mu);
P(1,1) = 1;                              % Weight on input 1
P(2,2) = 1;                              % Weight on input 2

Q = gen_q(Q1,P,N,M);
c = zeros(1,5*N);

%% Generate system matrixes for linear model
Aeq = gen_aeq(A1, B1, N, nx, mu);
beq = zeros(size(Aeq,1),1);
beq(1:nx) = A1*x0;

%% Solve QP problem with fmincon

f = @(z) 1/2*z'*Q*z;
opt = optimoptions('fmincon','Algorithm','sqp','MaxFunEvals', 4000);
tic

[z, zval, exitflag] = fmincon(f,z0,[], [], Aeq, beq, vlb, vub, @
   nl_constraints, opt);
toc

%% Extract control inputs and states
u1 = [z(N*nx+1:mu:n); z(n-1)]; % Control input from solution
u2 = [z(N*nx+2:mu:n); z(n)];

x1 = [x0(1);z(1:nx:N*nx)];               % State x1 from solution
x2 = [x0(2);z(2:nx:N*nx)];               % State x2 from solution
x3 = [x0(3);z(3:nx:N*nx)];               % State x3 from solution
x4 = [x0(4);z(4:nx:N*nx)];               % State x4 from solution
x5 = [x0(5);z(5:nx:N*nx)];               % State x5 from solution
x6 = [x0(6);z(6:nx:N*nx)];               % State x6 from solution

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

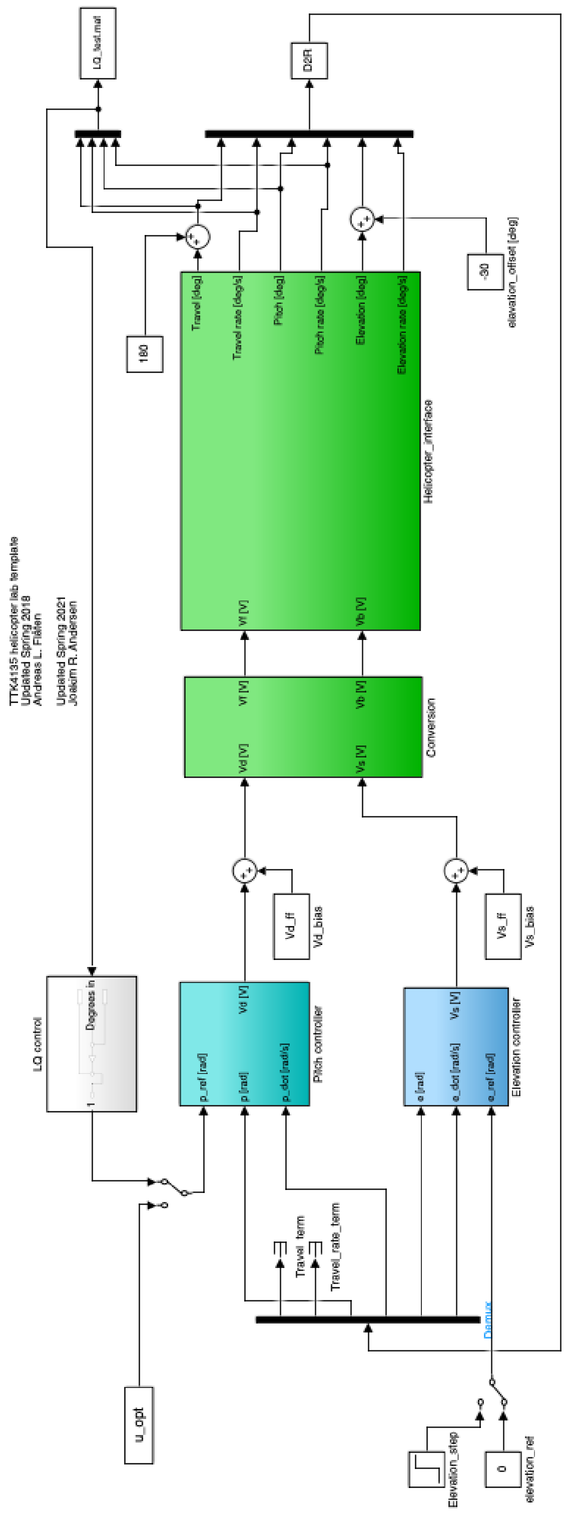
```

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];
```

B SIMULINK diagrams

B.1 LAB 2

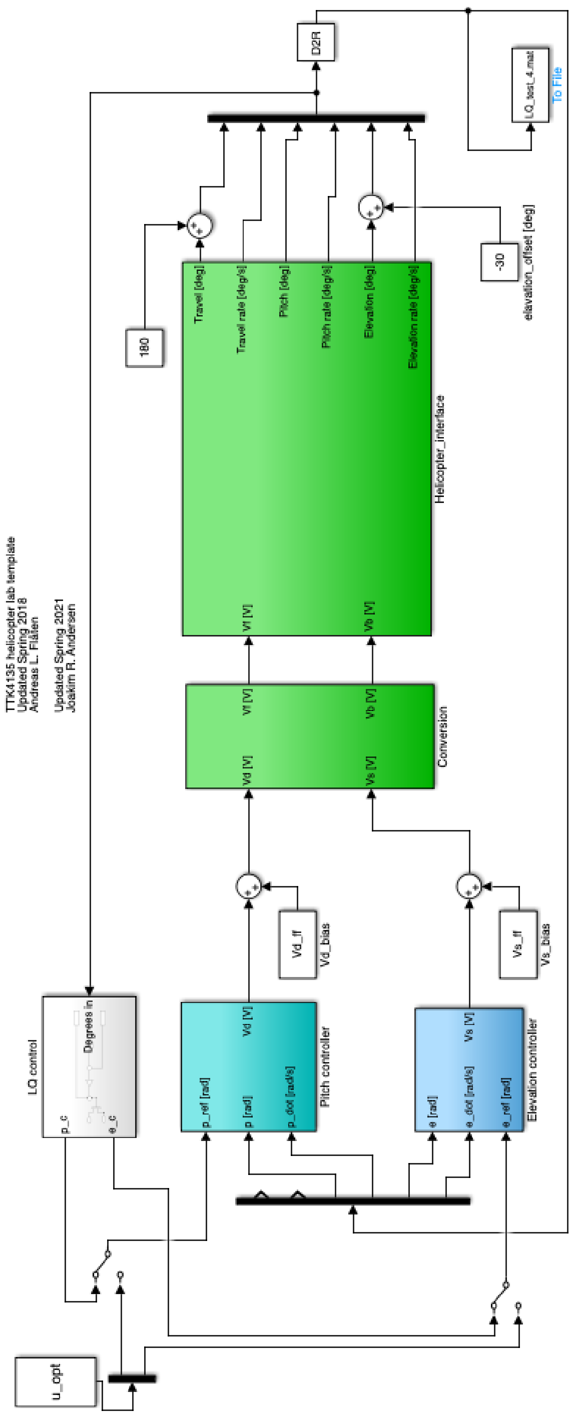


B.2 LAB 3



TTK4135 helicopter lab template
Updated Spring 2018
Andreas L. Fløtten
Updated Spring 2021
Joakim R. Andersen
LQ_test.mat
D2R
180
-30
elevation_offset [deg]
Travel [deg]
Travel rate [deg/s]
Pitch [deg]
Pitch rate [deg/s]
Elevation [deg]
Elevation rate [deg/s]
Vf [V]
Vb [V]
Helicopter_interface
Vd [V]
Vs [V]
Conversion
Vd_ff
Vd_bias
Vs_ff
Vs_bias
LQ control
Degrees in
p_ref [rad]
p [rad]
p_dot [rad/s]
Pitch controller
e [rad]
e_dot [rad/s]
e_ref [rad]
Elevation controller
Travel_term
Travel_rate_term
Demux
u_opt
Elevation_step
0
elevation_ref

B.3 LAB 4



TTK4135 helicopter lab template
Updated Spring 2018
Andreas L. Flåten
Updated Spring 2021
Joakim R. Andersen
LQ control
p_c
e_c
Degrees in
u_opt
p_ref [rad]
p [rad]
p_dot [rad/s]
Vd [V]
Pitch controller
e [rad]
e_dot [rad/s]
e_ref [rad]
Vs [V]
Elevation controller
Vd_ff
Vd_bias
Vs_ff
Vs_bias
Vd [V]
Vs [V]
Vf [V]
Vb [V]
Conversion
Vf [V]
Vb [V]
Travel [deg]
Travel rate [deg/s]
Pitch [deg]
Pitch rate [deg/s]
Elevation [deg]
Elevation rate [deg/s]
Helicopter_interface
180
-30
elevation_offset [deg]
D2R
LQ_test_4.mat
To File

References

[1] Norwegian University of Science and Technology Department of Engineering Cybernetics. TTK4135 Optimization and Control Helicopter Lab. 2023.

[2] Jan Dentler. *Real-time Model Predictive Control of Cooperative Aerial Manipulation.* PhD thesis, 10 2018.

[3] Andrzej Jezierski, Jakub Mozaryn, and Damian Suski. A comparison of lqr and mpc control algorithms of an inverted pendulum. 07 2017.

[4] N. Lehtomaki, N. Sandell, and M. Athans. Robustness results in linear-quadratic gaussian based multivariable control designs. *IEEE Transactions on Automatic Control*, 26(1):75–93, 1981.