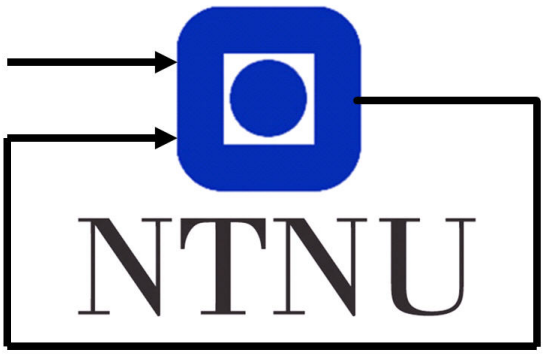
TTK4135 Optimization and Control
Lab Report

Group 29

Student 758034
Student 478254
Student 490111



Department of Engineering Cybernetics

Abstract

This report outlines algorithms for optimization and control of a stationary helicopter with three rotating joints, stabilized by low-level PD and PID controllers. The linearized model for the system dynamics are derived, and an open-loop optimal trajectory from the helicopters starting position to a final position is calculated by formulating a quadratic program with linear constraints. The open-loop is later closed by means of a linear quadratic controller, and results between open-loop and closed-loop control are discussed. Lastly, the effects of an additional nonlinear constraint is investigated.

Contents

1 Introduction

This report gives an introduction to optimization-based control of a rigidly mounted helicopter with three degrees of freedom, illustrating how optimization problems can be formulated and solved in the context of control. The implemented algorithms are specifically designed for the helicopter in the lab, but concepts and theory generalize to optimization in general, giving intuition to how optimization can be used in other real-world applications, and gives an introduction to some advantages and challenges that arise in optimization.

The report is organized as follows: Section 2 gives a description of the problem to be solved, the lab setup and the mathematical models describing the kinematics of the helicopter. Section 3 further introduces the optimization problem, optimizing a trajectory with linear constraints for the helicopter to follow without state feedback. Feedback is introduced in Section 4, correcting for deviations by a linear quadratic controller, with a subsequent discussion on the effects of feedback to the optimization. Lastly, an additional nonlinear constraint is added in Section 5, changing the optimal trajectory.

2 Problem Description

This section describes the laboratory setup and the mathematical model of the helicopter with low-level controllers, in addition to giving some physical parameters and variables used in the model.

2.1 Laboratory setup

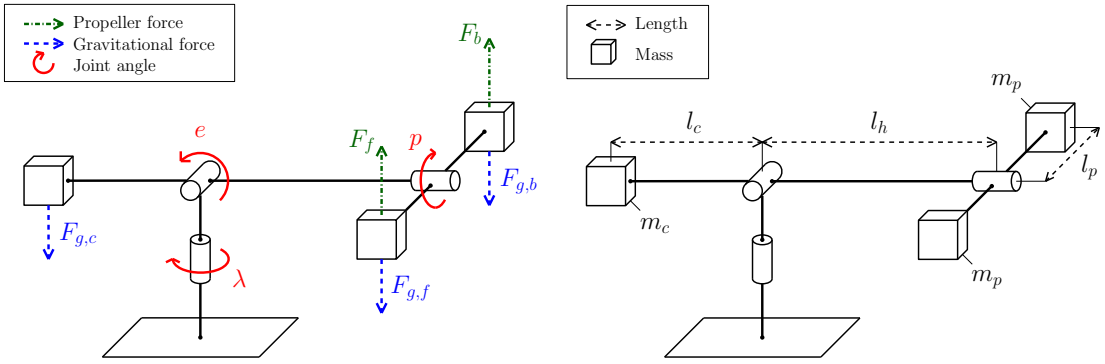
The laboratory setup, depicted in fig. 1, consists of a constrained helicopter with two rotors attached to a rotating joint, with the joint being part of a rotating arm with a counterweight at the opposite end. The arm can rotate in both horizontal and vertical directions, respectively denoted as the travel and elevation of the helicopter. The rotors may rotate perpendicular to the arm, here denoted as the pitch of the helicopter. Optical sensors are placed at each rotating joint, measuring the pitch, travel and elevation angle at all times, and one DC motor is placed on each rotor, generating an upward force that displaces the helicopter in the form of a pitch, travel and elevation angle. Relevant parameters and variables in the setup can be found in section 2.3.

2.2 Model

The system in fig. 1 can be described by the set of coupled ordinary differential equations for pitch p, elevation e and travel λ by using Newtons 2$^{\text{nd}}$ law of motion for rotation

$$J_\alpha \dot{\omega} = \sum \tau_\alpha = \sum F \cdot l_\alpha, \tag{1}$$

where J_α is the moment of inertia about some angle α, $\dot{\omega}$ is the angular acceleration, τ_α is the momentum about the angle α and F is the force applied at distance l_α from the joint angle α.



(a) Forces and angles in the model

(b) Masses and lengths in the model

Figure 1: Model of the helicopter (Image credit: Department of Engineering Cybernetics, NTNU)

Writing the forces in fig. 1a as $F_f = K_f V_f$ and $F_b = K_b V_b$, with the assumption that $K_f = K_b$, the equations of motion for elevation, pitch and travel may be derived in terms of parameters and variables in table 1 and table 2.

Elevation

In accordance to eq. (1) and fig. 1, the elevation is modeled as

$$J_e \ddot{e} = l_h K_f V_s - T_g, \tag{2}$$

such that

$$\ddot{e} = K_3 V_s - \frac{T_g}{J_e}, \quad K_3 = \frac{l_h K_f}{J_e}. \tag{3}$$

A PD controller is implemented to control the height, such that

$$V_s = K_{ep}(e_c - e) - K_{ed}\dot{e}, \quad K_{ep}, K_{ed} > 0, \tag{4}$$

leading to the system

$$\ddot{e} = -K_3 K_{ed}\dot{e} - K_3 K_{ep} e - \frac{T_g}{J_e} + K_3 K_{ep} e_c. \tag{5}$$

To account for deviations in the model, an integral term is added to the elevation controller, giving a PID controller. Assuming that the integral term counteracts the effect of $-\frac{T_g}{J_e}$, the resulting system for elevation is

$$\ddot{e} + K_3 K_{ed}\dot{e} + K_3 K_{ep} e = K_3 K_{ep} e_c. \tag{6}$$

Pitch

Following the derivation for elevation, the pitch angle is modeled by

$$J_p \ddot{p} = K_f l_p V_d, \tag{7}$$

giving

$$\ddot{p} = K_1 V_d, \quad K_1 = \frac{K_f l_p}{J_p}. \tag{8}$$

A PD controller is implemented to control the pitch angle, given as

$$V_d = K_{pp}(p_c - p) - K_{pd}\dot{p}, \quad K_{pp}, K_{pd} > 0, \tag{9}$$

leading to the closed-loop system

$$\ddot{p} + K_1 K_{pd}\dot{p} + K_1 K_{pp} p = K_1 K_{pp} p_c. \tag{10}$$

Travel

The travel is affected by the horizontal component of the forces, giving

$$J_t\dot{r} = -F_p l_h sin(p). \tag{11}$$

Note that $r = \dot{\lambda}$. A linear model is desired, so the system is linearized around $p = 0$, such that $sin(p) \approx p$, resulting in

$$\dot{r} = -K_2 p, \quad K_2 = \frac{F_p l_h}{J_t}. \tag{12}$$

To summarize, the linearized model of the helicopter is given by

$$\ddot{e} + K_3 K_{ed}\dot{e} + K_3 K_{ep} e = K_3 K_{ep} e_c, \tag{13a}$$
$$\ddot{p} + K_1 K_{pd}\dot{p} + K_1 K_{pp} p = K_1 K_{pp} p_c, \tag{13b}$$
$$\dot{\lambda} = r, \tag{13c}$$
$$\dot{r} = -K_2 p. \tag{13d}$$

The controllers for pitch and elevation are assumed tuned in the following sections, and will therefore not be further discussed.

2.3 Parameters and variables

Symbol	Parameter	Value	Unit
l_h	Distance from elevation axis to helicopter body	0.65	m
l_p	Distance from pitch axis to motor	0.17	m
l_c	Distance from elevation axis to counterweight	0.46	m
K_f	Force constant motor	0.0446	N/V
J_e	Moment of inertia for elevation	0.338	kg m^2
J_t	Moment of inertia for travel	0.338	kg m^2
J_p	Moment of inertia for pitch	0.0116	kg m^2
m_p	Mass of the motors	0.2	kg
m_c	Mass of counterweight	1.87	kg
m_g	Effective mass of the helicopter	0.03	kg
F_p	Force to lift the helicopter from the ground	0.2943	N

Table 1: Parameters and values.

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 horizontal

Table 2: Variables.

3 Optimal Control of Pitch/Travel without Feedback

In this part, we calculate the optimal trajectory x^* and corresponding input u^*, and use the sequence of setpoints as reference values to the inner controllers. There is no feedback from the states to the optimization layer, and deviations from the measured states will therefore not be corrected. For simplicity, we assume $e = 0$, i.e. the helicopter follows a horizontal trajectory.

3.1 State space model

The model in eq. (13) can be written on state space form, with $\mathbf{x} = [\lambda \quad r \quad p \quad \dot{p}]^\mathsf{T}$ as state-vector and $u = p_c$ as input, giving

$$\dot{\mathbf{x}} = \mathbf{A_c}\mathbf{x} + \mathbf{B_c}u, \tag{14}$$

where

$$\mathbf{A_c} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} \end{bmatrix}, \quad \mathbf{B_c} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1K_{pp} \end{bmatrix}.$$

The model includes four states and represents the physical and basic control layer shown in fig. 2, thus modeling both the helicopter and the low level controllers. As stated earlier, elevation is not included in our model, but linked to a constant reference such that the helicopter stays at a constant height. Consequently, the PID controller in the basic control layer will not be used in this part, only the PD controller.

3.2 Discretization

We discretize the model using the forward Euler method, given in state space form as

$$\frac{\mathbf{x}_{k+1} - \mathbf{x}_k}{\Delta t} = \mathbf{A_c}\mathbf{x}_k + \mathbf{B_c}u, \tag{15}$$

for some sampling time Δt. Solving for $\mathbf{x}_{k+1}$, we get

$$\mathbf{x}_{k+1} = \underbrace{(\mathbf{I} + \Delta t\mathbf{A}_c)}_{\mathbf{A}_d}\mathbf{x}_k + \underbrace{\Delta t\mathbf{B}_c}_{\mathbf{B}_d}\mathbf{u}_k, \tag{16}$$

as the discrete state space model, where the discretized system and input matrices are given by

$$\mathbf{A_d} = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & -K_2\Delta t & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & -K_1K_{pp}\Delta t & 1 - K_1K_{pd}\Delta t \end{bmatrix}, \quad \mathbf{B_d} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1K_{pp}\Delta t \end{bmatrix}.$$

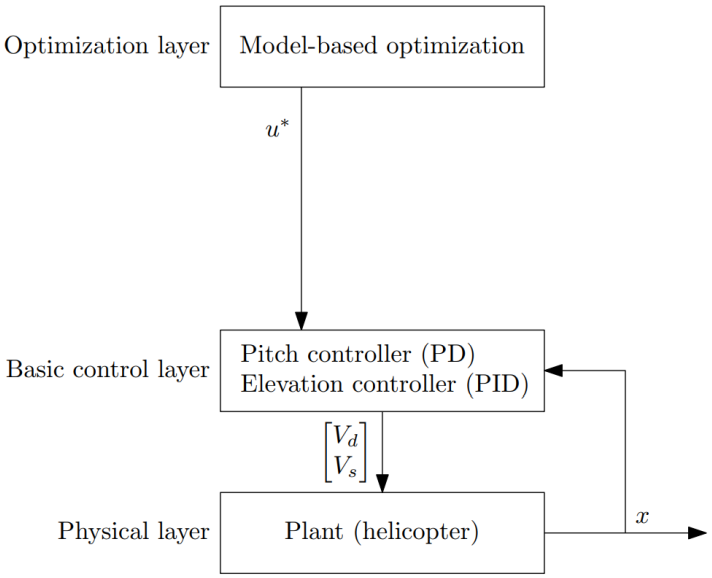


Figure 2: Illustration of the layers in the control hierarchy for this task (Image credit: Department of Engineering Cybernetics, NTNU)

3.3 Optimal trajectory and control

Now, we calculate the optimal trajectory for moving the helicopter from travel angle λ_0 to λ_f, that is, from $\mathbf{x}_0 = [\lambda_0 \quad 0 \quad 0 \quad 0]^\mathsf{T}$ to $\mathbf{x}_f = [\lambda_f \quad 0 \quad 0 \quad 0]^\mathsf{T}$. For the remainder of the report, we assume $\lambda_0 = \pi$ and $\lambda_f = 0$, that is, the helicopter should move 180° about the travel axis. The optimal trajectory will be constrained by

$$|p_k| \leq \frac{30\pi}{180}, \quad k \in \{1, \ldots, N\}, \tag{17}$$

which limits the pitch angle to an absolute value of 30°. The cost function to be minimized is given by

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

where N is the horizon to optimize over, p_{ci} is the setpoint for the p controller and q is a weight that penalizes the setpoints. To minimize the cost function, we use the `MATLAB` function `quadprog`[3], a solver for quadratic objective functions with linear constraints on the format

$$\min_x \frac{1}{2} \mathbf{z}^\top \mathbf{G}\, \mathbf{z} + \mathbf{c}^\top \mathbf{z} \quad \text{such that} \quad \begin{cases} \mathbf{A}\mathbf{z} \leq \mathbf{B} \\ \mathbf{A}_{eq}\mathbf{z} = \mathbf{B}_{eq} \\ \mathbf{lb} \leq \mathbf{z} \leq \mathbf{ub} \end{cases} \tag{19}$$

To utilize this function with our constraints, we need to manipulate both the cost function and our constraints, such that we may identify the desired matrices and vectors $\mathbf{G}$, $\mathbf{c}^\top$, $\mathbf{A}_d$, $\mathbf{B}_d$, $\mathbf{A}_{eq}$, $\mathbf{B}_{eq}$, $\mathbf{lb}$ and $\mathbf{ub}$. First, we look at eq. (16). To identify the equality constraints $\mathbf{A}_{eq}$ and $\mathbf{B}_{eq}$ we observe the behaviour as the steps iterate below

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{A}_c\mathbf{x}_0 + \mathbf{B}_c\mathbf{u}_0 \\ \mathbf{x}_2 &= \mathbf{A}_c\mathbf{x}_1 + \mathbf{B}_c\mathbf{u}_1 \\ &\vdots \\ \mathbf{x}_N &= \mathbf{A}_c\mathbf{x}_{N-1} + \mathbf{B}_c\mathbf{u}_{N-1} \end{aligned} \iff \begin{aligned} \mathbf{x}_1 - \mathbf{B}_c\mathbf{u}_0 &= \mathbf{A}_c\mathbf{x}_0 \\ \mathbf{x}_2 - \mathbf{A}_c\mathbf{x}_1 - \mathbf{B}_c\mathbf{u}_1 &= 0 \\ &\vdots \\ \mathbf{x}_N - \mathbf{A}_c\mathbf{x}_{N-1} - \mathbf{B}_c\mathbf{u}_{N-1} &= 0 \end{aligned} \tag{20}$$

Identifying the right-hand side of eq. (20) as $\mathbf{A}_{eq}\mathbf{z} = \mathbf{B}_{eq}$, where $\mathbf{z}$ is a vector containing states and inputs from the whole trajectory,

$$\mathbf{z} = [\ \underbrace{\lambda_1\ r_1\ p_1\ \dot{p}_1}_{\mathbf{x_1}}\quad \underbrace{\lambda_2\ r_2\ p_2\ \dot{p}_2}_{\mathbf{x_2}}\ \cdots\ \underbrace{\lambda_N\ r_N\ p_N\ \dot{p}_N}_{\mathbf{x}_N}\quad \underbrace{p_{c_0}\ p_{c_1}\quad \cdots\quad p_{c_{N-1}}}_{\mathbf{u}^*}\ ]^\mathsf{T}, \tag{21}$$

the matrices $\mathbf{A}_{eq}$ and $\mathbf{B}_{eq}$ may be written as

$$\mathbf{A}_{eq} = \begin{bmatrix} \mathbf{I} & 0 & \cdots & \cdots & 0 & -\mathbf{B} & 0 & \cdots & \cdots & 0 \\ -\mathbf{A} & -\mathbf{I} & \ddots & & \vdots & 0 & -\mathbf{B} & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 & \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\mathbf{A} & \mathbf{I} & 0 & \cdots & \cdots & 0 & -\mathbf{B} \end{bmatrix}, \tag{22}$$

$$\mathbf{B}_{eq} = \begin{bmatrix} \mathbf{A}_c\mathbf{x}_0 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{bmatrix}. \tag{23}$$

When rewriting the cost function eq. (18) in the general form of eq. (19), we notice that the linear term disappears due to $\lambda_f = 0$, such that $\mathbf{c}^\mathsf{T} = \mathbf{0}$. The cost function may then be written as

$$\phi = \frac{1}{2}\mathbf{z}^\mathsf{T}\mathbf{G}\mathbf{z} \tag{24}$$

$$= \frac{1}{2}\underbrace{(2{\lambda_1}^2\ +\ 2{\lambda_2}^2\ \cdots\ {\lambda_N}^2\ +\ 2q{p_{c_0}}^2\ \cdots\ 2q{p_{c_{N-1}}}^2)}_{\mathbf{z}^\top\mathbf{G}\,\mathbf{z}} \tag{25}$$

By examining $\mathbf{z}$, we see that $\mathbf{G}$ must be

$$\mathbf{G} = \begin{bmatrix} \mathbf{Q}_{(1)} & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \mathbf{Q}_{(N)} & \ddots & & \vdots \\ \vdots & & \ddots & 2q & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & 2q \end{bmatrix}, \tag{26}$$

where

$$\mathbf{Q}_{(\,\cdot\,)} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{27}$$

Lastly, we identify $\mathbf{lb}$ and $\mathbf{ub}$, which are solely limited by eq. (17). The states that are not influenced by the constraint have bounds set to $-\infty$ and ∞, such that

$$\begin{aligned} \mathbf{lb} =& [\ [-\infty\ -\infty\ -\frac{30\pi}{180}\ -\infty]_1^\mathsf{T}\ \cdots\ [-\infty\ -\infty\ -\frac{30\pi}{180}\ -\infty]_N^\mathsf{T}\ ]^\mathsf{T}, \\ \mathbf{ub} =& [\ [\infty\ \infty\ \frac{30\pi}{180}\ \infty]_1^\mathsf{T}\ \cdots\ [\infty\ \infty\ \frac{30\pi}{180}\ \infty]_N^\mathsf{T}\ ]^\mathsf{T}. \end{aligned} \tag{28}$$

In the assignment, several helper-functions are handed out in order to calculate some of the large matrices described so far. These functions are:

- `gen_q`: Generates $\mathbf{G}$
- `gen_aeq`: Generates $\mathbf{A}_{eq}$
- `gen_constraints`: Generates $\mathbf{lb}$ and $\mathbf{ub}$
- `diag_repeat`: Produces a diagonal of the matrix input

Now, the desired matrices and vectors are known, and with the help of `quadprog` we plot the manipulated variable and the output, seen in fig. 3. The simulations are done with three different values of q: 0.1, 1, and 10. A large q value penalizes the input and makes it expensive. On the contrary, low q values makes input values cheap and results in a faster and more aggressive behaviour. In this case, the parameter penalizing the state error in the cost function is constant equal to 1. From our plots in fig. 3 we see the following results:

$q = \mathbf{0.1}$**:** As predicted, the input reaches it's upper and lower limitations quickly and has a aggressive behaviour. This is because input is cheap. Consequently, the pitch is quite fast as well, because the low q results in a more penalized state error. However, because of the overshooting it may be too aggressive, because it may create unwanted wear and tear.

$q = \mathbf{1}$: Less aggressive input, but still quite fast. The pitch response has a less oscillatory response, and the state error and input are punished equally much.

$q = \mathbf{10}$: Now, we punish high input values the most. The input reaches it's limits, but are smoother and way less aggressive than with the lower q values. The input is now punished more than the state error, but even though the pitch response is less aggressive, the helicopter reaches its travel destination almost as fast as with the other q values.

Lastly, we will take a closer look at the cost function in eq. (18), and the unwanted effects that may appear when λ_i reaches λ_f. When that happens, the function solely consists of qp_{ci}^2, thus, making p_{ci}^2 become 0. With the speed and momentum the helicopter may have at $\lambda_f = 0$, the helicopter may pass the endpoint, and as a result not being at the correct travel point when the input sequence completes. This is explained further in section 3.4.

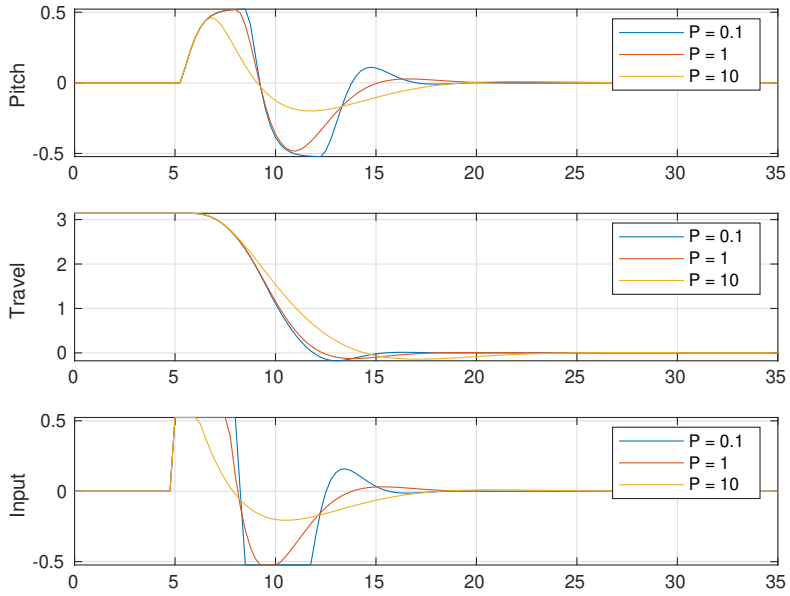


Figure 3: The travel, pitch and optimal input, u*, calculated with three different values of q: 1, 0.1 and 10.

3.4 Optimal trajectory in practice

The helicopter is now simulated with the implemented open-loop controller in `SIMULINK`, depicted in appendix B.1, with `MATLAB` code in appendix A.1, and run with help from QuaRC and Realtime Workshop. To allow the helicopter to stabilze before the optimal

trajectory is fed into the controllers, a large sequence of zeros is padded at the beginning and end of the optimal trajectory. The calculated setpoints are fed into the pitch controller as desired reference values. Figure 4 shows the calculated path in pitch, in addition to the measured paths with $q = 0.1$, 1 and 10, while fig. 5 shows the travel. One can clearly see that the helicopter struggles to reach its calculated travel path, especially at as the travel moves towards $\lambda_f = 0$. The reason for this is as stated earlier in section 3.3, the models lack of taking into account the travel rate and momentum of the helicopter as it reaches $\mathbf{x}_f$, the input u is set to zero.

Whereas there is a deviation between the optimal travel rate and the calculated travel rate shown in fig. 6, which is calculated to be 0 at $\mathbf{x}_f$ and is clearly not the case for the measured values. This discrepancy between the calculated control variable and what is needed to slow down the helicopter in order to reach and maintain $\mathbf{x}_f$, causes the travel to move beyond $\mathbf{x}_f$.

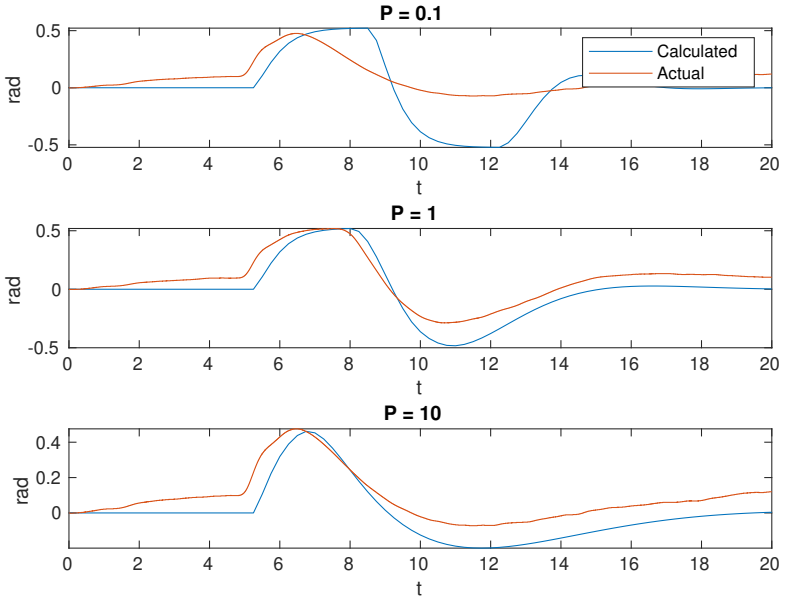


Figure 4: Calculated trajectory of pitch vs. actual trajectory

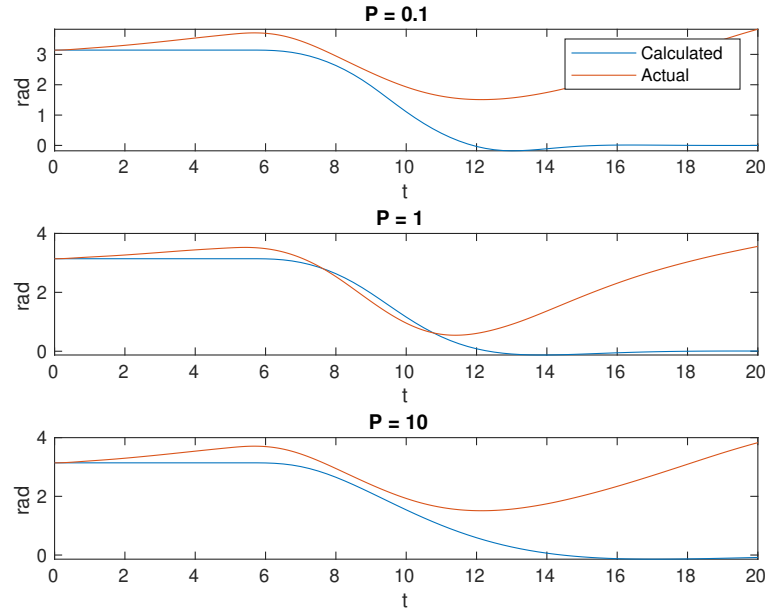


Figure 5: Calculated trajectory of travel vs. actual trajectory

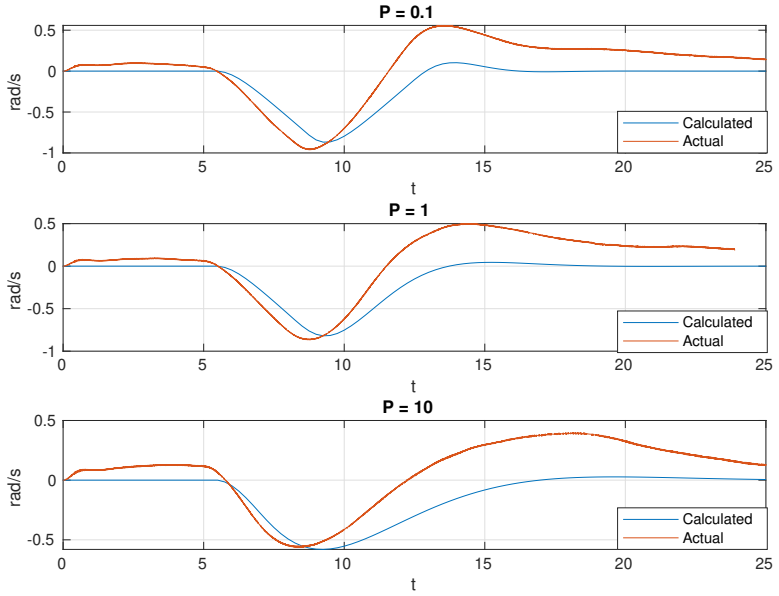


Figure 6: Calculated trajectory of travel rate vs. actual trajectory

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

This section introduces feedback in the form of a linear quadratic (LQ) controller, often denoted as a linear quadratic regulator (LQR), improving upon some of the difficulties discovered in section 3.

4.1 LQ control

In order to achieve a better response to the calculated flight path from the optimization problem, state feedback is implemented to account for deviations in the model, with the new control hierarchy shown in fig. 7.

The input to the system is now a function of the system's state, given as

$$u_k = u_k^* - \mathbf{K}^\mathsf{T}(\mathbf{x}_k - \mathbf{x}_k^*), \tag{29}$$

for some gain $\mathbf{K}$. The optimal $\mathbf{K}$ is found by minimizing the quadratic objective function

$$J = \sum_{i=0}^{\infty} \Delta\mathbf{x}_{i+1}^\top \mathbf{Q} \Delta\mathbf{x}_{i+1} + \Delta u_i^\top R \Delta u_i, \qquad \mathbf{Q} \geq 0, \quad R > 0, \tag{30}$$

with minimum found analytically by the algebraic Riccati equation. In `MATLAB` this can be done with the function `dlqr`[1]. In eq. (30), we have $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}^*$ and $\Delta u = u - u^*$, such that $\mathbf{Q}$ and R penalize deviations from the optimal trajectory and input. A crucial task is then to choose the weights that give best performance. As both $\mathbf{Q}$ and R affect the performance similarly, R is kept at a constant $R = 1$ for the remainder of the report, and will thus not limit the input u. $\mathbf{Q}$ is then tuned to give optimal performance.

4.2 Tuning and LQR in practice

The `SIMULINK` diagram for the new control hierarchy is shown in appendix B.2, with the addition of the `MATLAB` code in appendix A.2 to the code in appendix A.1. The different values of $\mathbf{Q}$ will be given as the diagonal elements,

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

where q_1, q_2, q_3 and q_4 penalize deviations in travel, travel rate, pitch and pitch rate, respectively. The pitch and travel recorded from the helicopter using different values for $\mathbf{Q}$ is shown in fig. 8. Our tuning strategy started by tuning q_1 which represent the weight for travel, and when travel follows the calculated path adequately, we moved over to q_2, q_3 and q_4. It is clear to see that the feedback greatly improves the helicopters ability to follow the calculated path. It can also be seen in fig. 9 that the travel rate more closely follows the calculated path than for the implementation where no feedback is used. The

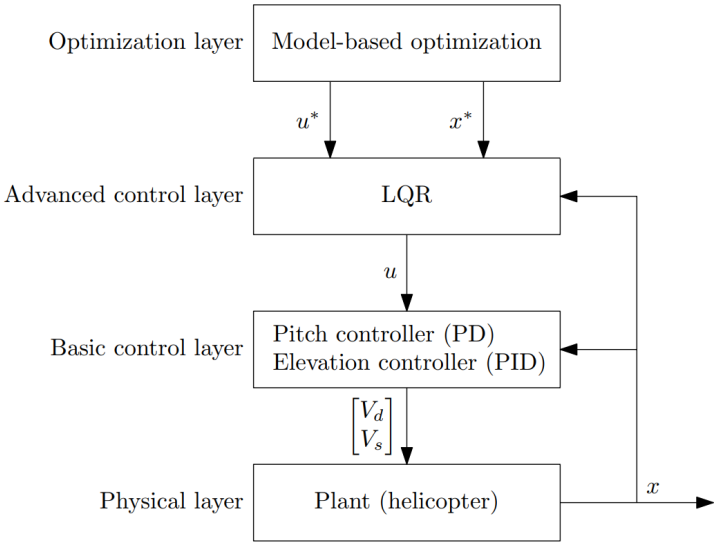


Figure 7: Illustration of the layers in the control hierarchy for this task (Image credit: Department of Engineering Cybernetics, NTNU)

focus here has been on the optimal trajectory for travel, which is why penalties in the deviation of pitch is relaxed.

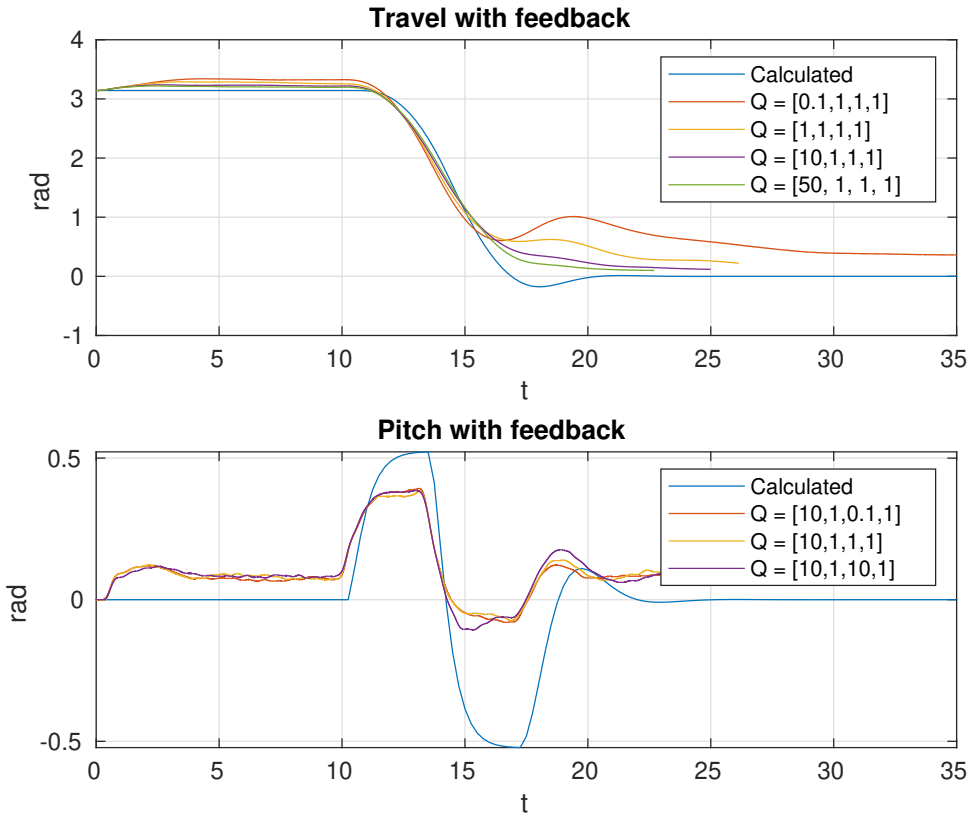


Figure 8: Pitch and travel with feedback implemented

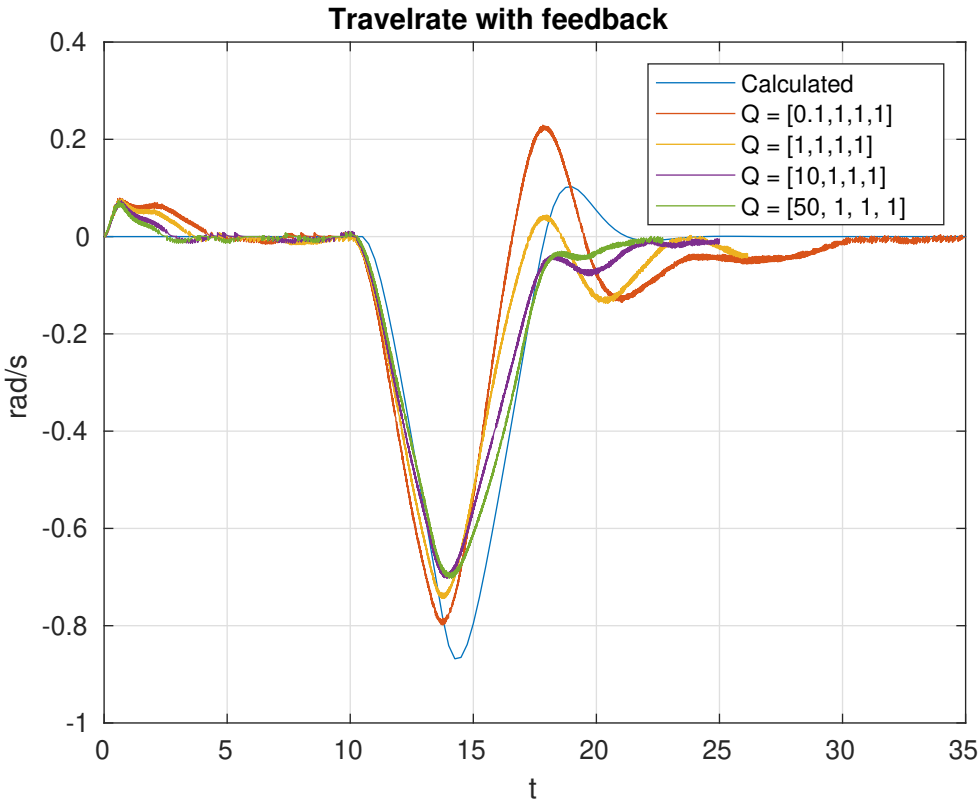


Figure 9: Travel rate with feedback implemented

4.3 Model Predictive Control (MPC)

As stated in the problem text, one option to improve the overall performance of the system could be to implement an MPC. An MPC can be seen as a model based optimization, where, for each controller timestep, the model is updated with the latest measurements and the optimization problem is re-calculated. The first output from the solved optimization problem is then used as the setpoint of lower level controllers on the system, until the process is re-run for each MPC iteration.

This approach has the advantage that the optimization problem is run with the latest measured values, providing better feedback than solutions tried in this project, where the entire optimization is calculated before running the system. This being the main difference between a MPC and a LQR based control approach, a LQR optimizes for a set time horizon and does not update the control output based on measured inputs, where as the MPC does that by solving the optimization problem for each iteration. This gives the difference between LQR and MPC concerning fig. 7, that the MPC layer would give setpoints directly to the inner loop controllers, instead of the optimal calculated path from the optimization problem passed though the LQR layer. The MPC effectively functions as the model-based optimization and LQR layer.

One downside when using the MPC approach, is the added calculation time due to the need to solve an optimization problem for each iteration of the higher-level controller, which may be very time consuming depending on the time horizon.

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

This section extends the optimization to two dimensions, where the helicopter as before moves from an initial point to a reference point with a constraint on pitch, but now also with an additional nonlinear constraint on elevation, acting as an obstacle in the original trajectory. The state space model is first extended and discretized, before the nonlinear constraint is introduced with a subsequent discussion of the effects on the performance of the helicopter.

5.1 State space model

The state space model is extended to include elevation and elevation rate, as the elevation changes due to the constraint. In continuous time, with state-vector $\mathbf{x} = [\lambda \quad r \quad p \quad \dot{p} \quad e \quad \dot{e}]^\mathsf{T}$, it takes the form

$$\dot{\mathbf{x}} = \mathbf{A_c}\mathbf{x} + \mathbf{B_c}\mathbf{u}, \tag{31}$$

with

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

$$\mathbf{B_c} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3K_{ep} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} p_c \\ e_c \end{bmatrix},$$

where the equation for elevation in eq. (13) is used for the two additional states.

5.2 Discretization

The discretization of eq. (31) follows the forward Euler method in section 3.2, giving

$$\mathbf{x}_{k+1} = \mathbf{A_d}\mathbf{x}_k + \mathbf{B_d}\mathbf{u}, \tag{32}$$

where

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

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

5.3 Optimization with nonlinear inequality constraint on elevation

The helicopter will as before follow an optimal trajectory from an initial point $\mathbf{x}_0$ to a reference point $\mathbf{x}_f$, now subject to the additional inequality constraint in elevation given as

$$e_k \geq \alpha \exp(-\beta(\lambda_k - \lambda_t)^2) \quad \forall k \in \{1, \ldots, N\}, \tag{33}$$

with $\alpha = 0.2$, $\beta = 20$ and $\lambda_t = \frac{2\pi}{3}$. A possible constraint is visualized in fig. 10, where the helicopter must keep an elevation angle above the drawn graph from the initial point to the reference at all times. The QP solver utilized in the previous tasks presupposes linear constraints, so a SQP-type algorithm is instead used here as the constraint on elevation is nonlinear. In `MATLAB` this is done with the function `fmincon`[2], taking in the nonlinear constraint defined in the function `nonlincon` in appendix A.3.

The new criteria to be minimized is now

$$\phi = \sum_{i=0}^{N-1} (\lambda_{i+1} - \lambda_f)^2 + q_1 p_{ci}^2 + q_2 e_{ci}^2, \tag{34}$$

where elevation is penalized by q_2. Setting the reference travel angle to $\lambda_f = 0$, the criteria can be written in the general form

$$\phi = \frac{1}{2}\mathbf{z}^\mathsf{T}\mathbf{G}\mathbf{z}, \tag{35}$$

with

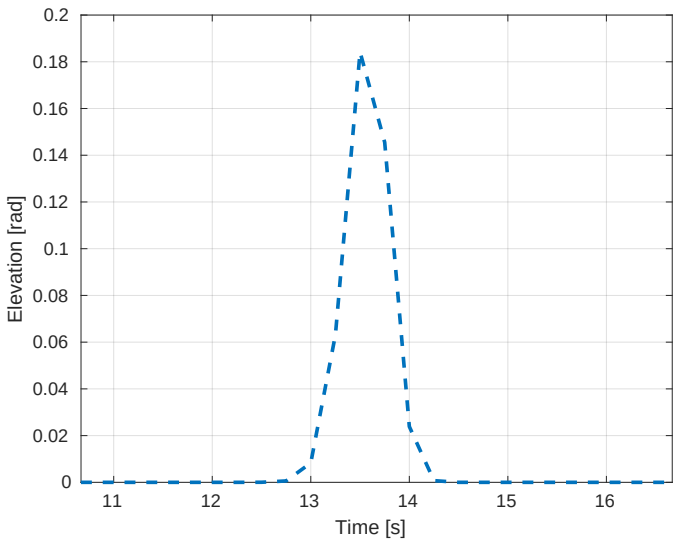


Figure 10: Example of the inequality constraint in eq. (33) with $\alpha = 0.2$, $\beta = 20$ and $\lambda_t = \frac{2\pi}{3}$.

$$\mathbf{z} = \begin{bmatrix} \mathbf{x}_1^\mathsf{T} & \dots & \mathbf{x}_N^\mathsf{T} & \mathbf{u}_0^\mathsf{T} & \dots & \mathbf{u}_{N-1}^\mathsf{T} \end{bmatrix}^\mathsf{T},$$

$$\mathbf{G} = \begin{bmatrix} \mathbf{Q} & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \mathbf{Q} & \ddots & & \vdots \\ \vdots & & \ddots & \mathbf{R} & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & \mathbf{R} \end{bmatrix},$$

where

$$\mathbf{Q} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} 2q_1 & 0 \\ 0 & 2q_2 \end{bmatrix}.$$

The remaining part of the optimization is to define the lower and upper bounds for the states, $\mathbf{lb}$ and $\mathbf{ub}$, and the equality constraint $\mathbf{A_{eq}z} = \mathbf{b_{eq}}$, which is done as in

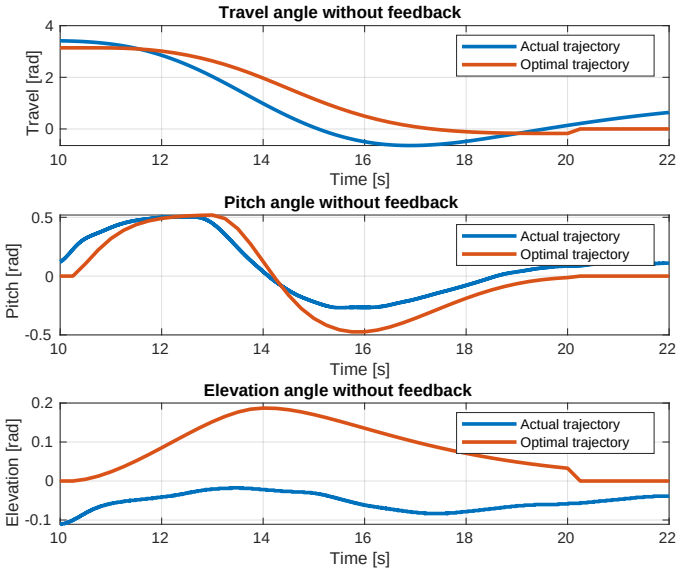


Figure 11: Optimal travel, pitch and elevation vs. measured without feedback

section 3. The full `MATLAB` code for this section is found in appendix A.3, while the corresponding `SIMULINK` diagram is found in appendix B.3.

5.4 Open-loop optimal control

The performance of an open-loop optimal controller is first investigated, with results shown in fig. 11. The helicopter follows the optimal trajectory in travel and pitch decently, but can be seen to drift off at $t \approx 20$s due to the open loop. This is especially significant for travel, which continues to drift, by the same reason as in section 3. The helicopter fails drastically to keep up with the optimal trajectory in elevation, breaking the inequality constraint put on elevation. This illustrates the need for feedback, as an optimal controller that is unable to obey its constraints ultimately fails in its task. This is especially crucial for real-world applications. The poor performance of the open-loop controller does however not come as a surprise, as we know the equations of motion in eq. (13) do not capture all the dynamics in the model.

5.5 Closed-loop optimal control

Closing the loop with an LQR as in section 4, the performance of the helicopter is expected to improve. Following the procedure for tuning discussed in section 4, the weights chosen for the LQR in the closed-loop system were found to be

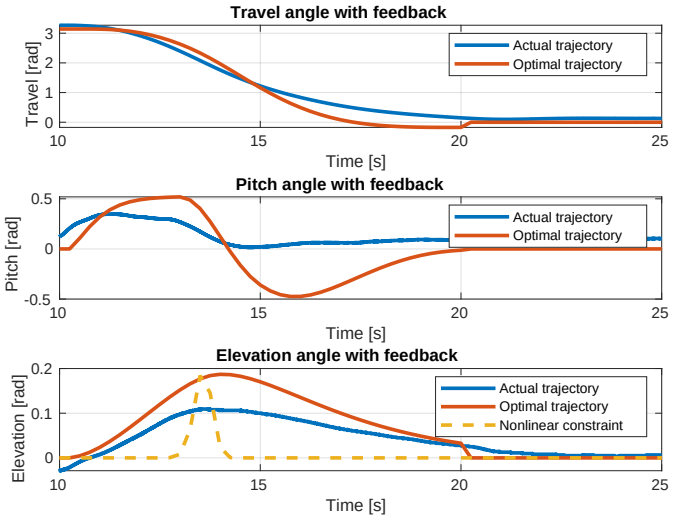


Figure 12: Optimal travel, pitch and elevation vs. measured with feedback

$$\mathbf{Q}_{\text{LQR}} = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 40 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R}_{\text{LQR}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The main focus here isn't tuning the LQR, so these weights are not optimal in any way, but give satisfactory performance. The results can be seen in fig. 12. Comparing to fig. 11, we see that travel and especially elevation are significantly improved, while pitch has worsened. Although there may exist configurations for $\mathbf{Q}_{\text{LQR}}$ and $\mathbf{R}_{\text{LQR}}$ that achieve good performance in all states, it is seemingly difficult to achieve without having to sacrifice one or more states. In this case, pitch has been sacrificed in favor of travel and elevation. This is due to the specific task in the lab, where we prioritize a helicopter that follows an optimal travel trajectory while obeying the constraint in elevation. As the constraint should be upheld, deviations in elevation were penalized heavier than deviations in travel. An analysis similar to that in section 3 was done in regard to the weights on the input in the optimization, q_1 and q_2, arriving at $q_1 = 1$ and $q_2 = 1$.

As seen from the plot of elevation in fig. 12, the helicopter doesn't uphold the nonlinear constraint. An attempt to fix this was to penalize deviations in elevation even harder, while relaxing the weight on travel, but the helicopter could still not maintain the optimal trajectory in elevation. This may be due to imperfections in the model, which may be corrected for by integral effect in the LQR. Looking at fig. 12, we can indeed see that the

actual trajectory has an offset from the optimal, even though it should have stabilized at $e = 0°$ due to the zero padding. The performance may thus be improved by introducing integral effect, or by further tuning. Worth noting is also that the model is only valid around the linearization point, which may explain deviations from the optimal trajectory when the helicopter is outside equilibrium.

5.6 Model imperfections

As briefly mentioned, the model of the helicopter is simplified and does not capture all the dynamics in the system. Due to the linearization when deriving eq. (13), the first 4 states are completely decoupled from the last 2. In reality, elevation and pitch are coupled states. This is especially apparent when the pitch is close to 90°, as there won't be any upward thrust to increase elevation. The states are tighter coupled when the pitch increases, as there won't be generated as much upwards thrust, and the model ultimately underestimates the elevation of the helicopter. As the elevation constraint peaks near the mid-turn, where the pitch is highest, the elevation depletes and will not be able to uphold the nonlinear constraint. With the added offset from a steady-state error, due to lack of integral effect in the LQR, it is difficult to tune a controller that obeys the constraint.

When deriving the equation of motion for elevation, an assumption of $p \approx 0$ is implicitly made, as the true equation is $J_e\ddot{e} = l_h K_f V_s cos(p) - T_g$. For small angles, the approximation $cos(p) \approx 1$ holds, giving the linearized model used throughout the lab. When the optimal trajectory is calculated without taking the true dynamics into account, the trajectory may request excessive pitch angles causing control of the elevation and elevation rate to suffer. This means that changes in pitch might cause deviations from the optimal trajectory without this being correctly penalized by the controllers.

An improvement to correct for the model imperfections is naturally to work with nonlinear dynamics. This would change the control hierarchy of the system, as for example an LQR would no longer work. Due to lack of experience with nonlinear systems, it is difficult to say how this change would affect other parts of the optimization. Another possible improvement would be to add stricter constraints so that the approximations hold better. For example, a stricter constraint on pitch would make the approximations $sin(p) \approx p$ and $cos(p) \approx 1$ hold better. In an attempt to correct for the model imperfections, the constraint on pitch was set to $|p_k| \leq \frac{15\pi}{180}$, giving a smaller pitch throughout the trajectory. Due to the decrease in pitch angle, the helicopter was seen to obey the constraint on elevation slightly better, but still not good enough. Additionally, as travel is coupled with pitch, the helicopter moved too slowly to its reference point, making the need to increase the horizon of the optimization.

6 Conclusion

In this report, a helicopters flight is controlled using setpoints given from an optimization problem using a linearized model. Two different cases are discussed, one with linear constraints and one with an additional nonlinear constraint. For both of these cases, feedback using a LQR approach are investigated against an open-loop approach. From this two clear conclusions can be drawn, firstly, feedback performs much better for controlling the helicopter towards it's calculated path. Secondly, model discrepancies causes especially travel and elevation control to be sub-optimal, especially when elevation constraints are introduced. This leads to the recommendation that for improving this design a MPC using a better, likely unlinear model is introduced. An approach that can also be investigated further is the usage of harder constraints on the pitch variable, as this is seen to have an influence on the elevation control.

A MATLAB Code

A.1 Section 3

```
%% Initialization and model definition
init08;

delta_t = 0.25;                          % sampling time

% State vector: x = [lambda r p p_dot]'
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]';

Ad = Ac*delta_t  + eye(4);
Bd = Bc*delta_t;


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

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

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

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

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

```
xl(3)   = ul;                           % Lower bound on state x3
xu(3)   = uu;                           % Upper bound on state x3

% Generate constraints on measurements and inputs
[vlb,vub]       = gen_constraints(N, M, xl, xu, ul, uu);
vlb(N*mx+M*mu)  = 0;                    % We want the last input to be zero
vub(N*mx+M*mu)  = 0;                    % We want the last input to be zero

% Generate the matrix Q and the vector c
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 = 20;                               % Weight on input
Q = gen_q(Q1, P1, N, M);               % Generate Q
c = [];                                % Generate c

%% Generate system matrices for linear model
Aeq = gen_aeq(Ad, Bd, N, mx, mu);      % Generate Aeq
beq = zeros(400,1);                    % Generate beq
beq(1:4) = Ad*x0;

%% 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];

%% Generate data for Simulink
t = 0:delta_t:delta_t*(length(u)-1);
u_opt = [t', u];
```

A.2 Section 4

```
%% LQR
Q = diag([10 1 1 1]); % [travel, travel_rate, pitch, pitch_rate]
R = 1;                % pitch_ref
K = dlqr(Ad, Bd, Q, R);
```

A.3 Section 5

```
function [c, ceq] = nonlincon(z)
    N = 40;
    lambda_k = z(1:6:N*6);
    e_k = z(5:6:N*6);
    alpha = 0.2;
    beta = 20;
    lambda_t = 2*pi/3;

    c = alpha * exp(-beta * (lambda_k - lambda_t).^2) - e_k;
    ceq = [];
end
```

```
%% Initialization and model definition
init08;

delta_t = 0.25;                         % sampling time

% State vector: x = [lambda r p p_dot e e_dot]'
```

```
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];

Ad = Ac*delta_t  + eye(size(Ac,1));
Bd = Bc*delta_t;


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

% Initial values
x1_0 = pi;                                % Lambda
x2_0 = 0;                                 % r
x3_0 = 0;                                 % p
x4_0 = 0;                                 % p_dot
x5_0 = 0;                                 % e
x6_0 = 0;                                 % e_dot
x0 = [x1_0 x2_0 x3_0 x4_0 x5_0 x6_0]';  % Initial values

% 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 G and the vector c
G1 = zeros(mx,mx);
G1(1,1) = 2;                            % Weight on state x1
G1(2,2) = 0;                            % Weight on state x2
G1(3,3) = 0;                            % Weight on state x3
G1(4,4) = 0;                            % Weight on state x4
G1(5,5) = 0;                            % Weight on state x5
G1(6,6) = 0;                            % Weight on state x6
P1 = 2*diag([1, 1]);                    % Weight on input
G = gen_q(G1, P1, N, M);                % Generate G
c = [];                                 % Generate c

% LQR
Q = diag([10 1 1 1 1 1]);
R = diag([1 1]);
K = dlqr(Ad, Bd, Q, R);

% Generate system matrixes for linear model
Aeq = gen_aeq(Ad, Bd, N, mx, mu);       % Generate A
beq = zeros(size(Aeq,1),1);             % Generate b
beq(1:6) = Ad*x0;

%% Solve SQP problem with nonlinear constraint
fun = @(z) 0.5*z'*G*z;
options = optimoptions('fmincon');
options.MaxFunEvals = 40000;
tic
[z,lambda] = fmincon(fun, z0, [], [], Aeq, beq, vlb, vub, @nonlincon, 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+G(i,i)*z(i)*z(i);
```

```
  PhiOut(i) = phi1;
end

%% Extract control inputs and states
u  = z(N*mx+1:N*mx+M*mu);% Control input from solution
u1 = u(1:mu:N*mu);
u2 = u(2:mu:N*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)];              % state x5 from solution
x6 = [x0(6);z(6:mx:N*mx)];              % state x6 from solution

num_variables = 10/delta_t;
zero_padding = zeros(num_variables,1);
unit_padding  = ones(num_variables,1);

u1  = [zero_padding; u1; zero_padding; 0];
u2  = [zero_padding; u2; zero_padding; 0];
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];


%% Generate data for Simulink
t = 0:delta_t:delta_t*(length(u1)-1);

u1_opt = [t', u1];
u2_opt = [t', u2];
x1_opt = [t', x1];
x2_opt = [t', x2];
x3_opt = [t', x3];
x4_opt = [t', x4];
x5_opt = [t', x5];
x6_opt = [t', x6];

u_opt = [t', u1, u2];
x_opt = [t', x1, x2, x3, x4, x5, x6];
```

B Simulink Diagrams

B.1 Section 3

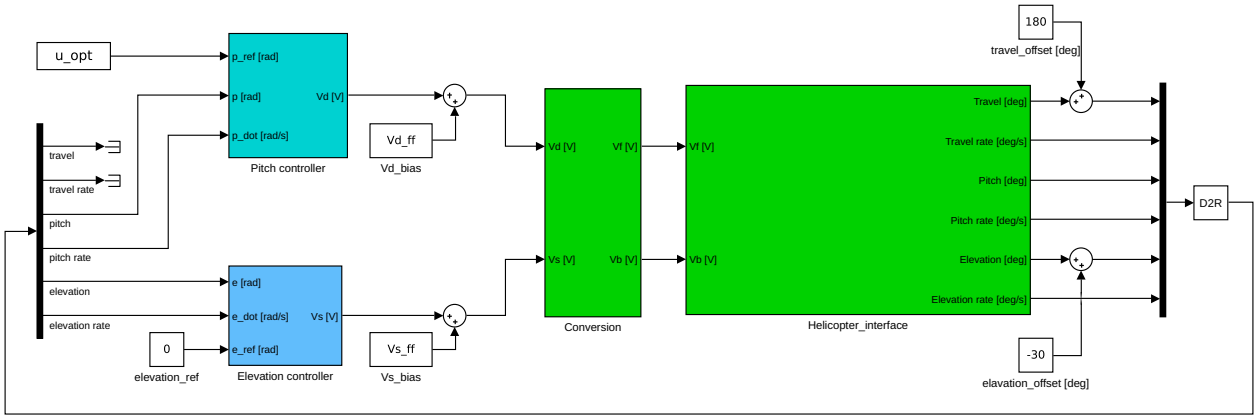


Figure 13: Simulink model from section 3.

B.2 Section 4

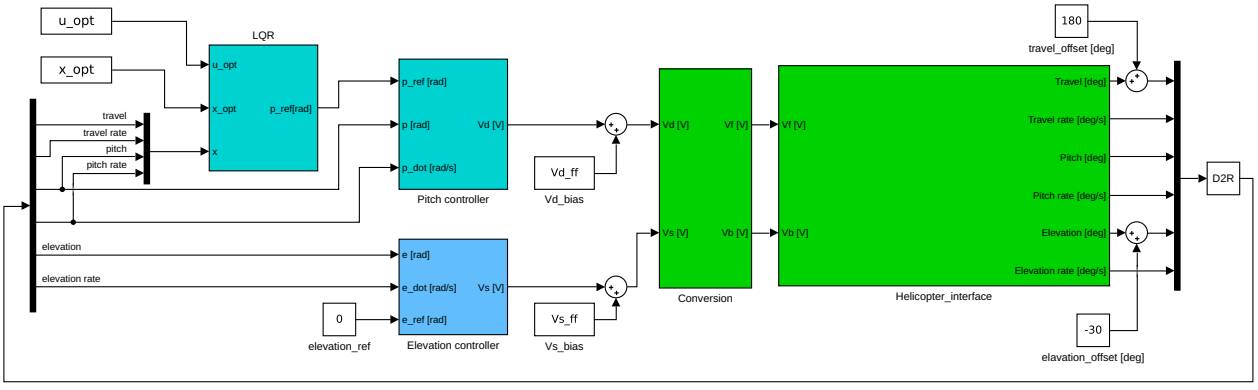


Figure 14: Simulink model from section 4.

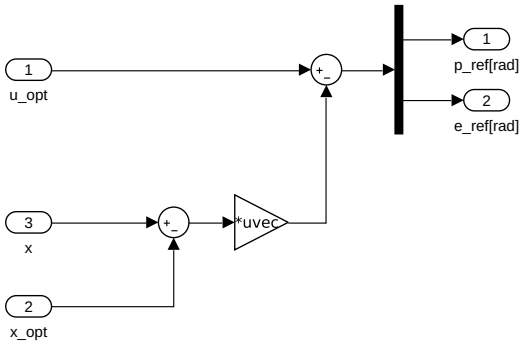


Figure 15: Simulink model of LQR from section 4.

B.3 Section 5

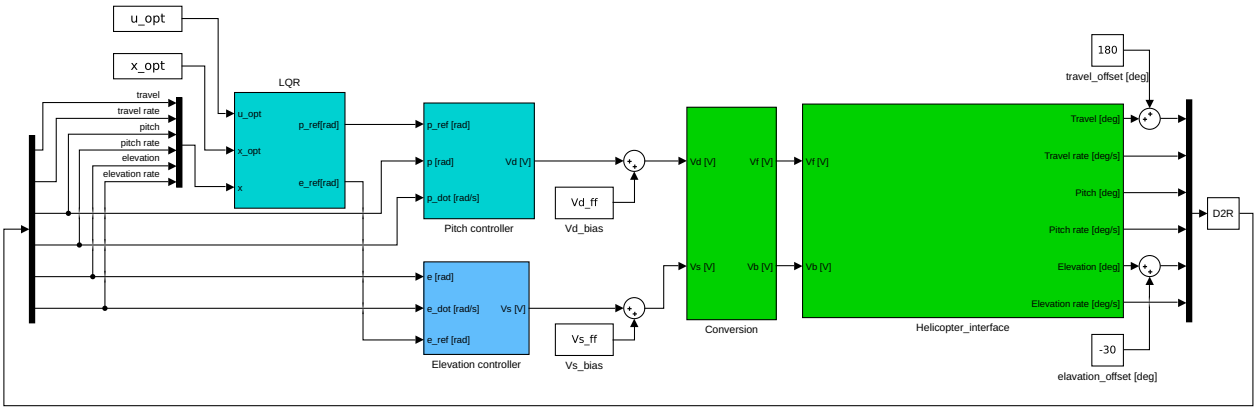


Figure 16: Simulink model from section 5.

References

[1] *MATLAB documentation for function* `dlqr`. `https://se.mathworks.com/help/control/ref/dlqr.html`. Accessed: 2020-03-01.

[2] *MATLAB documentation for function* `fmincon`. `https://se.mathworks.com/help/optim/ug/fmincon.html`. Accessed: 2020-03-01.

[3] *MATLAB documentation for function* `quadprog`. `https://se.mathworks.com/help/optim/ug/quadprog.html`. Accessed: 2020-03-01.