Optimization and control

Magnus Dyre-Moe
Eivind Heldal Stray
Anders Breistøl

Department of Engineering Cybernetics

Abstract

The objective of this project is to solve an optimization problem and then implement it on a helicopter system. The result of this is an optimal input sequence that is fed into the helicopter in order for it to follow a specified trajectory. To achieve this, three different solutions have been executed: travel control with no feedback, travel control with feedback and a combination of travel and elevation control with feedback. The project was divided into three days, one for each solution.

Contents

1 Introduction

In this project, optimization was to be used to calculate an optimal trajectory from one position to another. In order for the helicopter to be able to follow this trajectory, an optimal input sequence had to be calculated, also using optimization. This was then implemented on the helicopter.

The assigned tasks were split into three different exercises:

In the first exercise, found in section 3, the intention was to make the helicopter follow a 180° horizontal trajectory. This was to be done by calculating an optimal trajectory x^*, and a corresponding optimal input sequence u^*. The elevation angle was to be assumed constant $e = 0$, and the measured state was not to be fed back to correct deviations during the helicopters flight.

In the second exercise, found in section 4, the intention was the same as part 1, making the helicopter follow a 180° horizontal trajectory. This time though, feedback was to be introduced in the optimal controller, in order to correct deviations during the helicopters flight. This was to be done using a Linear Quadratic (LQ) controller.

In the third exercise, found in section 5, the intention was to make the helicopter follow a trajectory that involved a 180° turn with a restriction on elevation. This essentially meant that the helicopter was to climb and descend during its 180°turn, meaning elevation was to be controlled in addition to travel and pitch. In this task, both open-loop and closed-loop control was to be used.

1.1 Perspective

In all three exercises, different limitations were put on input and degrees of movement to assure the motors would not be overloaded and the solutions were realistic. The methods of control implemented in this report can be used in robotic arms, airplanes, cars and a lot of other systems that require limitations in movements and actions as well as a calculation of optimal solutions for simple or complex problems.

1.2 Structure

This report is structured in the following order: The problem description in section 2 includes a model of the helicopter, the lab setup and a quick summary of physical modeling of the helicopter. The three exercises, section 3, section 4 and section 5 are split into the associated sub-exercises, with a discussion and presentation of results at the end of each exercise. The conclusion is found in section 6, and includes a short summary of this lab as well as closing remarks. Matlab code used in this lab are to be found in appendix A, and the highest level Simulink diagram in appendix B. The bibliography can be found on page 33.

Throughout this report, any variable presented with bold text is to be considered a matrix or vector, unless otherwise specified.

2 Problem description

The lab setup consist of an on-lab computer which communicates with the lab helicopter through QuaRC. QuaRC integrates seamlessly with Simulink, which allows Simulink-models to be run in real-time.

The helicopter itself consists of a base connected to an arm. The arm has the helicopter attached to one side, and a counterweight connected to the opposite side. The helicopter side consist of two motors creating lift, which can move 180°around the axis of the arm. This is illustrated in Figure 1.

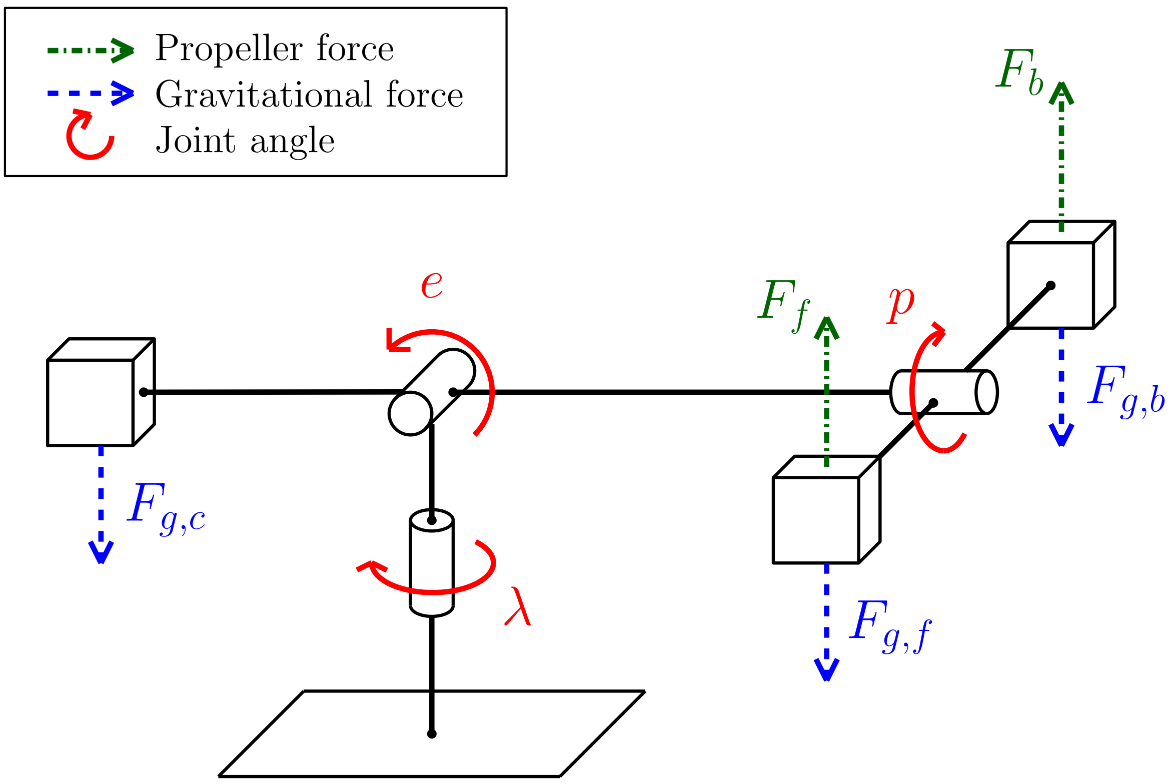


Figure 1: Helicopter setup with forces[1]

2.1 Physical modeling

The physical modeling of the helicopter can be summarized as shown in the following equations (1)[2].

$$\ddot{e} + K_3 K_{ed}\dot{e} + K_3 K_{ep} e = K_3 K_{ep} e_c \tag{1a}$$
$$\ddot{p} + K_1 K_{pd}\dot{p} + K_1 K_{pp} p = K_1 K_{pp} p_c \tag{1b}$$
$$\dot{\lambda} = r \tag{1c}$$
$$\dot{r} = -K_2 p \tag{1d}$$

[1]downloaded from report-guide
[2]Copied from delivered code

2.2 Parameters, values and variables

The following parameters, values and variables will be used throughout this report. Table 1 shows all the general values for the type of helicopter used in this project. In reality, some of the values may be slightly different due to wear and tear or other factors.

Table 1: Parameters and values[3].

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 m^2
J_t	Moment of inertia for travel	0.83	kg m^2
J_p	Moment of inertia for pitch	0.034	kg m^2
m_h	Mass of helicopter	1.05	kg
m_w	Balance weight	1.87	kg
m_g	Effective mass of the helicopter	0.05	kg
K_p	Force to lift the helicopter from the ground	0.49	N

Table 2: Variables[4].

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

[3]Copied from delivered code

[4]Copied from assignment text

3 Exercise 2 - Optimal control for pitch/travel without feedback

3.1 Part 1 - State space for continuous time

For simplicity in the lab report the input, $\boldsymbol{u}$, will be written in bold font even though the input is scalar in Section 3 and 4.

The goal of this task is to model the helicopter in continuous time, on the form:

$$\dot{\boldsymbol{x}} = \boldsymbol{A}_c\boldsymbol{x} + \boldsymbol{B}_c\boldsymbol{u} \tag{2}$$

with state

$$\boldsymbol{x} = \begin{bmatrix}\lambda & r & p & \dot{p}\end{bmatrix}^T \tag{3}$$

and input

$$\boldsymbol{u} = p_c \tag{4}$$

The physical modeling of the helicopter in the assignment text can be found in equation (1). Combining the equations of motion with the desired state $\boldsymbol{x}$ renders the resulting state-space equation:

$$\begin{bmatrix}\dot{\lambda}\\ \dot{r}\\ \dot{p}\\ \ddot{p}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & 0 & -K_2 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd}\end{bmatrix}\begin{bmatrix}\lambda\\ r\\ p\\ \dot{p}\end{bmatrix} + \begin{bmatrix}0\\ 0\\ 0\\ K_1K_{pp}\end{bmatrix}p_c \tag{5}$$

This model includes travel, travel rate, pitch and pitch rate in that particular order. Viewing the system in regard to Figure 2 the model contains the physical layer and the basic control layer in form of the physical dimensions of the helicopter and the PD-regulator used for control. A separate controller handles the elevation, as specified by the assignment. In the model, the helicopters response to different voltage inputs V_s and V_d, is considered.

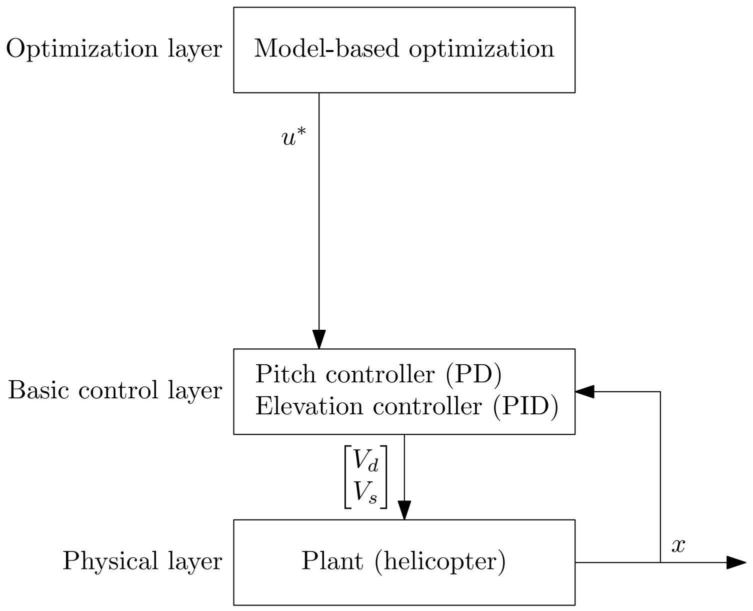


Figure 2: Illustration of the layers in the control hierarchy for an open-loop process[5]

[5]Copied from assignment text

3.2 Part 2 - State space for discrete time

The forward Euler method is as following:

$$\frac{\boldsymbol{x}_{k+1} - \boldsymbol{x}_k}{\Delta t} = \boldsymbol{A}_c \boldsymbol{x}_k + \boldsymbol{B}_c \boldsymbol{u}_k \tag{6}$$

Solving the equation results in

$$\begin{aligned} \boldsymbol{x}_{k+1} &= (\boldsymbol{I} + \Delta t \boldsymbol{A}_c)\boldsymbol{x}_k + \Delta t \boldsymbol{B}_c \boldsymbol{u}_k \\ &= \boldsymbol{A}_d \boldsymbol{x}_k + \boldsymbol{B}_d \boldsymbol{u}_k \end{aligned} \tag{7}$$

With

$$\boldsymbol{A}_d = \boldsymbol{I} + \Delta t \boldsymbol{A}_c = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & -\Delta t K_2 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & -\Delta t K_1 K_{pp} & 1 - \Delta t K_1 K_{pd} \end{bmatrix} \tag{8}$$

and

$$\boldsymbol{B}_d = \Delta t \boldsymbol{B}_c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \Delta t K_1 K_{pp} \end{bmatrix} \tag{9}$$

Using $\Delta t = 0.25$ means

$$\boldsymbol{A}_d = \begin{bmatrix} 1 & 0.25 & 0 & 0 \\ 0 & 1 & -0.14 & 0 \\ 0 & 0 & 1 & 0.25 \\ 0 & 0 & -0.81 & 0.10 \end{bmatrix} \tag{10}$$

has the eigenvalues $\lambda_{1,2,3,4} = 1, 1, 0.55, 0.55$ calculated by $eig(\boldsymbol{A}_d)$ in matlab. Hence, the system is said to be marginally stable in discrete time. The discrete $\boldsymbol{B}$-matrix will, with a time-step $\Delta t = 0.25$, is

$$\boldsymbol{B}_d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.81 \end{bmatrix} \tag{11}$$

The ctrb function in Matlab was used to check the system for controllability [1]:

```
sys = ctrb(A_d, B_d)
rank(sys)


ans =

     4
```

Thus the controllability matrix, $\boldsymbol{\mathcal{C}} = [\boldsymbol{B} \quad \boldsymbol{AB} \quad \boldsymbol{A}^2\boldsymbol{B} \quad \boldsymbol{A}^3\boldsymbol{B}]$ has full rank. Thus, the system is controllable, and every state can be controlled.

Clearly the system is observable, since every position state can be read directly. From each position over time, the rate of the corresponding state can be calculated.

3.3 Part 3 - Optimization problem

For simplicity in the lab report the $\boldsymbol{R}$-matrix will written in bold font even though $\boldsymbol{R}$ is scalar in Section 3 and 4.

In this part, an optimal trajectory for moving the helicopter from $\boldsymbol{x}_0 = [\lambda_0 \quad 0 \quad 0 \quad 0]^T$ to $\boldsymbol{x}_f = [\lambda_f \quad 0 \quad 0 \quad 0]^T$ will be calculated, with the assumption that the elevation angle is constant. The implemented constraint is, in this case, $|p_k| \leq \frac{30\pi}{180}$, k$\in \{1, ..., N\}$. The cost function (12) also needs to be minimized.

$$\phi = \sum_{i=0}^{N-1} (\lambda_{i+1} - \lambda_f)^2 + qp_{ci}^2 \quad , \quad q \geq 0 \tag{12}$$

In order to implement this in Matlab, the optimization problem needs to be formulated as a standard QP problem. A standard QP problem is on the form:

$$\min_{\boldsymbol{z}} f(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{G}\boldsymbol{z} + \boldsymbol{c}^T\boldsymbol{z} \quad s.t \quad \begin{cases} \boldsymbol{A}_{eq}\boldsymbol{z}^* = \boldsymbol{B}_{eq} \\ \boldsymbol{A}_{ineq}\boldsymbol{z}^* \geq \boldsymbol{B}_{ineq} \\ \boldsymbol{z}^* \geq \boldsymbol{0} \end{cases} \tag{13}$$

Because the optimization problem only has a quadratic term and no inequality constraint and a lower and upper bound, the QP problem can be written as:

$$\min_{\boldsymbol{z}} f(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{G}\boldsymbol{z} \quad s.t \quad \begin{cases} \boldsymbol{A}_{eq}\boldsymbol{z}^* = \boldsymbol{B}_{eq} \\ \boldsymbol{z}_{high} \geq \boldsymbol{z}^* \geq \boldsymbol{z}_{low} \end{cases} \tag{14}$$

Even though there is no inequality constraint there is equality constraint which will be addressed shortly. The cost function can be then be reformulated as

$$\begin{aligned} \boldsymbol{\phi} &= \sum_{i=0}^{N-1} \frac{1}{2}(\boldsymbol{x}_{i+1} - \boldsymbol{x}_0)^T\boldsymbol{Q}(\boldsymbol{x}_{i+1} - \boldsymbol{x}_0) + \frac{1}{2}\boldsymbol{u}_i^T\boldsymbol{R}\boldsymbol{u}_i \\ &= \sum_{i=0}^{N-1} \frac{1}{2}\boldsymbol{x}_{i+1}^T\boldsymbol{Q}\boldsymbol{x}_{i+1} + \frac{1}{2}\boldsymbol{u}_i^T\boldsymbol{R}\boldsymbol{u}_i \end{aligned} \tag{15}$$

with

$$\begin{aligned} \boldsymbol{Q} &\geq \boldsymbol{0} \\ \boldsymbol{R} &\geq \boldsymbol{0} \end{aligned} \tag{16}$$

Evaluating the cost function in regard to $\boldsymbol{Q}$ and $\boldsymbol{R}$ it is possible to observe that in order to deliver λ, the following is needed:

$$\boldsymbol{Q} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad , \quad \boldsymbol{R} = q \tag{17}$$

At this point, the QP problem in question needs be written on the form as seen in (14). The $\boldsymbol{z}$ vector is a combination of states and inputs over a time horizon N. More specifically $\boldsymbol{z}$ is of dimension (N*mx + N*mu) x 1 where mx is the number of states and mu is the number of inputs

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_N \\ \boldsymbol{u}_0 \\ \vdots \\ \boldsymbol{u}_{N-1} \end{bmatrix} \tag{18}$$

Because of the nature of (14) the $\boldsymbol{G}$ matrix becomes a matrix of (N*mx + N*mu) x (N*mx + N*mu) dimensions. The $\boldsymbol{G}$ matrix then becomes

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{Q}_1 & \mathbf{0} & \cdots & \cdots & \cdots & \mathbf{0} \\ \mathbf{0} & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \boldsymbol{Q}_N & \ddots & & \vdots \\ \vdots & & \ddots & \boldsymbol{R}_0 & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \cdots & \cdots & \mathbf{0} & \boldsymbol{R}_{N-1} \end{bmatrix} \tag{19}$$

Where the $\boldsymbol{Q}$ portion of the matrix is of (N*mx x N*mx) dimensions and the $\boldsymbol{R}$ portion is of (N*mu x N*mu) dimensions.

Equation (7) can be written on the form

$$-\boldsymbol{A}_d\boldsymbol{x}_k + \boldsymbol{x}_{k+1} - \boldsymbol{B}_d\boldsymbol{u}_k = 0 \tag{20}$$

for every step of the simulation. This is illustrated in (21)

$$\begin{aligned} -\boldsymbol{A}_d\boldsymbol{x}_0 + \boldsymbol{x}_1 - \boldsymbol{B}_d\boldsymbol{u}_0 &= 0 \\ -\boldsymbol{A}_d\boldsymbol{x}_1 + \boldsymbol{x}_2 - \boldsymbol{B}_d\boldsymbol{u}_1 &= 0 \\ &\vdots \\ -\boldsymbol{A}_d\boldsymbol{x}_{N-1} + \boldsymbol{x}_N - \boldsymbol{B}_d\boldsymbol{u}_{N-1} &= 0 \end{aligned} \tag{21}$$

The result of this is the two following matrices

$$\boldsymbol{A_{eq}} = \left[\begin{array}{ccccc|ccccc} \boldsymbol{I} & \mathbf{0} & \cdots & \cdots & 0 & -\boldsymbol{B}_d & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ -\boldsymbol{A}_d & \boldsymbol{I} & \ddots & & \vdots & \mathbf{0} & \ddots & \ddots & & \vdots \\ \mathbf{0} & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \mathbf{0} & \vdots & & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & -\boldsymbol{A}_d & \boldsymbol{I} & \mathbf{0} & \cdots & \cdots & \mathbf{0} & -\boldsymbol{B}_d \end{array}\right] \tag{22}$$

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

By implementing the equations and matrices to matlab, the problem can be solved using the matlab function quadprog:

```
quadprog(G,c,[],[],Aeq,Beq,vlb,vub,z0)
```

Where $\boldsymbol{G}$, $\boldsymbol{A}_{eq}$, $\boldsymbol{B}_{eq}$ are described earlier. $\boldsymbol{vlb}$ and $\boldsymbol{vub}$ is the lower and upper bound constraint in (14), $\boldsymbol{z}_0$ is the initial position of the helicopter and $\boldsymbol{c}$ is a vector of zeros.

Because travel is dependant of pitch the optimization problem will create a sequence of pitch values for the helicopter in order to reach the desired trajectory.

Now the optimization layer illustrated in Figure 2 has been designed.

More detailed implementation of the optimization layer is displayed in appendix A.4.

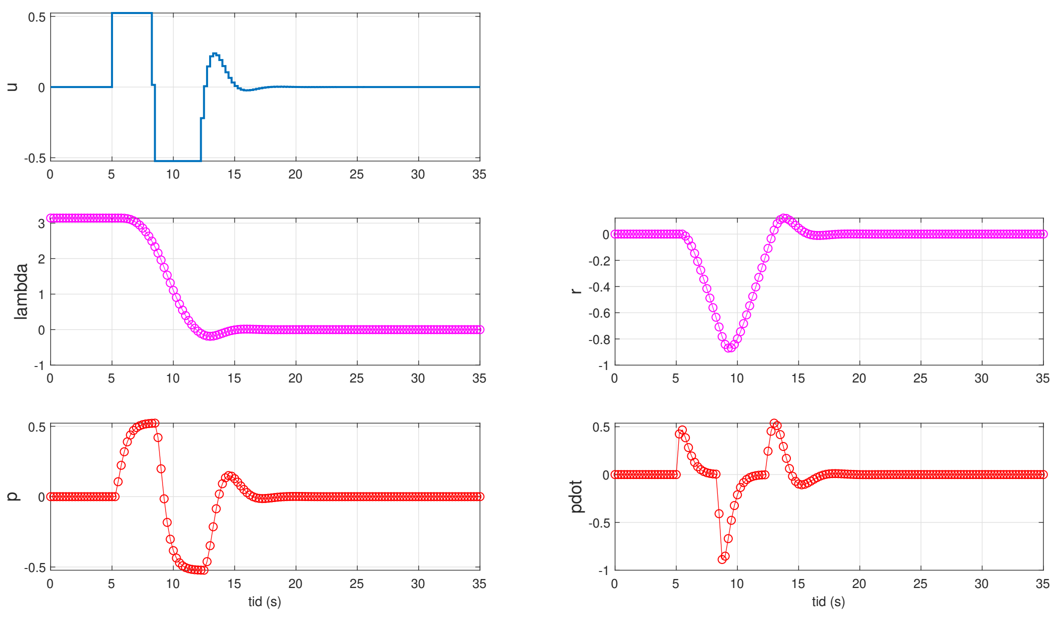


Figure 3: Simulated states with q = 0.1

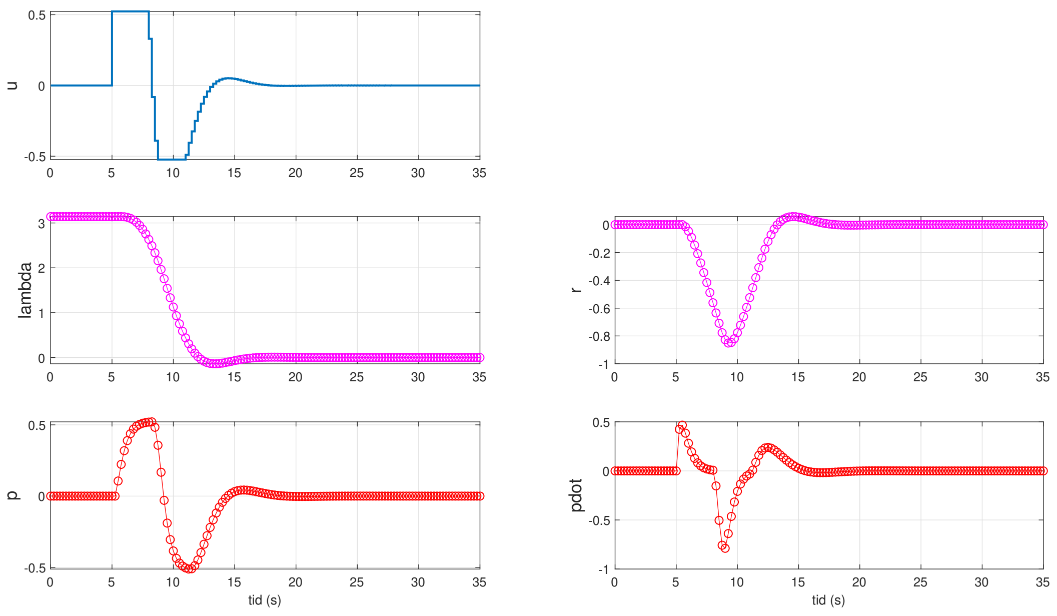


Figure 4: Simulated states with q = 1

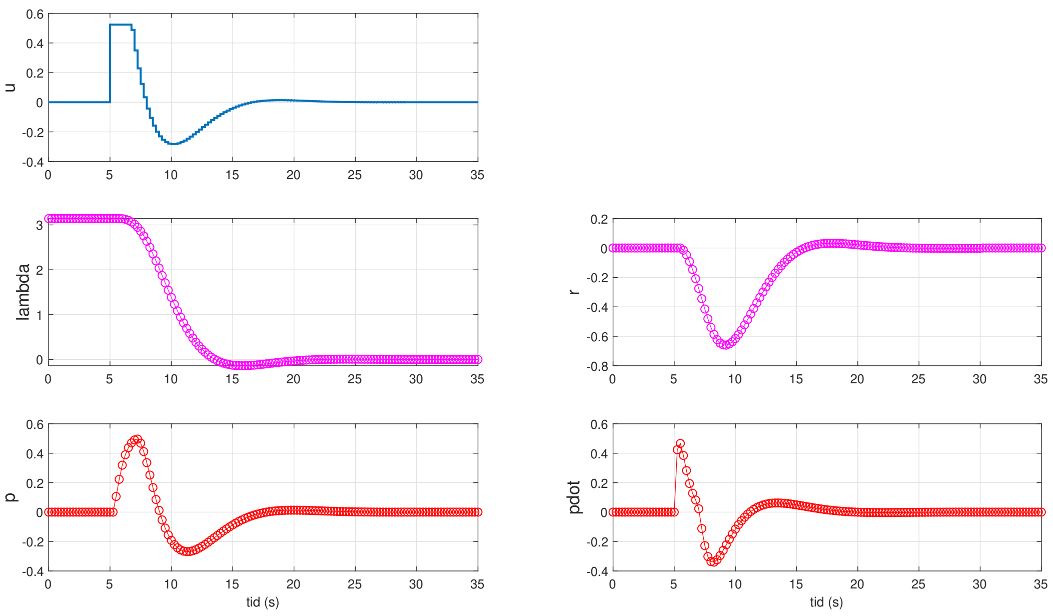


Figure 5: Simulated states with q = 10

3.3.1 Discussion

Looking at the plots in Figure 3, Figure 4 and Figure 5, a great difference in both the simulated states and the input voltage can be observed. The higher q is, the smoother the contours of the curves are. This is not surprising as q is a weight on the input voltage. When q is weighted heavier, it is given a higher restriction. The input becomes steady with slow change, and compared to lower values for q, there is a drastic change in the behaviour of the input. The restriction put on the input voltage is then easy to observe from the simulated states with $\dot{p}$ displaying the largest change from a low to a high q.

As of now, only simulated input and states have been used to drive the helicopter. When doing so the helicopter is supposed to follow a set pattern given to the helicopter via $from\ workspace$ in simulink. In the simulation the helicopter reaches its travel-reference. However, because of small irregularities and a simplified model the same is not certain for the actual helicopter. Once the simulation reaches reference the input controlling pitch goes to zero. However, the helicopter itself does not necessarily reach the reference. Also, the helicopter has a tendency to drift when the motors are perfectly horizontal. Since the controller does not have state feedback for travel, an offset in travel and travel rate is not compensated. Thus, the helicopter may continue to drift in addition to ending up somewhere else than intended. This is easy to correct by implementing a state-feedback as shown in the following exercise.

3.4 Part 4 - Simulation vs actual trajectory without feedback

Since this system is open loop, the trajectory is not updated as it is executed. Thus, the helicopter does not act on deviations from the optimal trajectory, making it subject to noise and errors. The pitch plotted in Figure 6 follows a reference quite good, but the travel in Figure 7 does not. Because travel is not corrected, and is also dependant on pitch to adjust deviations from optimal trajectory (which is not updated in open loop), the helicopter has no way of making up for errors.

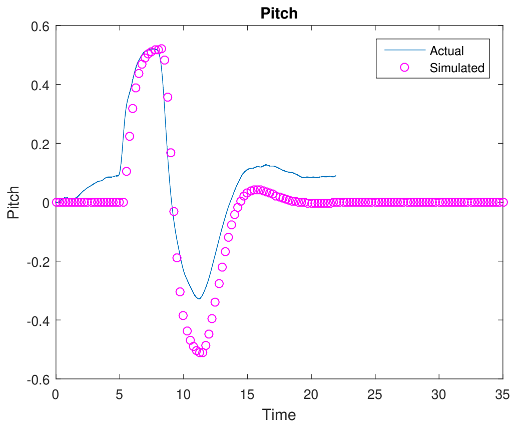


Figure 6: Simulated pitch and actual pitch

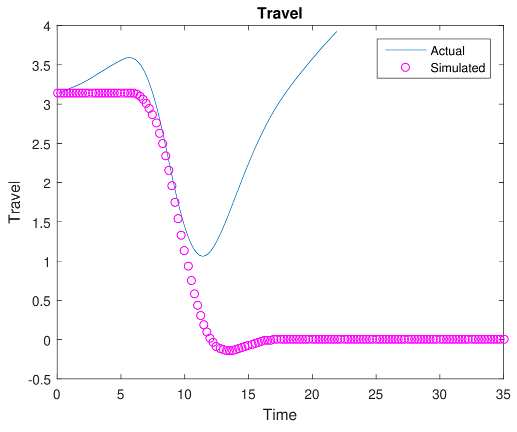


Figure 7: Simulated travel and actual travel

3.5 Results and conclusion for the day

In this exercise, we have solved the optimization problem for travel with different values for q. This provided us with a sequence $\boldsymbol{u} = [\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_{N-1}]^T$ that was fed into the helicopter without state feedback. Since this solution is open-loop, the helicopters accuracy was not great, and it ended up drifting after it was meant to stay still. However, it is safe to say the helicopter did go in the desired direction, and somewhat close to the goal position, π radians. From 6 and 7 one can see that the actual pitch deviates from the desired pitch, but not that much. In addition, the travel drifts before and after the maneuver. The maneuver itself, although not far from 0 to π, is imperfect. This is a direct consequence of the fact that our model is imperfect in addition to unforeseen noise such as draft form the windows in the lab, and non-linear effects.

4 Exercise 3 - Optimal control of pitch/travel with feedback (LQ)

4.1 Part 1 - LQ-controller

In this section, the goal is to implement feedback to the optimal controller by use of an LQ controller, using the same $\boldsymbol{x}^*$ and $\boldsymbol{u}^*$ calculated in section 3. For an LQ controller, $\boldsymbol{u}_k = \boldsymbol{u}_k^* - \boldsymbol{K}^T(\boldsymbol{x}_k - \boldsymbol{x}_k^*)$. The following cost function also needs to be minimized:

$$\boldsymbol{J} = \sum_{i=0}^{\infty} \Delta\boldsymbol{x}_{i+1}^T \boldsymbol{Q}\boldsymbol{x}_{i+1} + \Delta\boldsymbol{u}_i^t \boldsymbol{R}\Delta\boldsymbol{u}_i \quad , \quad \boldsymbol{Q} \geq \boldsymbol{0}, \boldsymbol{R} \geq \boldsymbol{0} \tag{24}$$

where

$$\Delta\boldsymbol{x}_{i+1} = \boldsymbol{A}\Delta\boldsymbol{x}_i + \boldsymbol{B}\Delta\boldsymbol{u}_i \tag{25}$$

and

$$\Delta\boldsymbol{x}_i = \boldsymbol{x}_i - \boldsymbol{x}_i^* \text{ and } \Delta\boldsymbol{u}_i = \boldsymbol{u}_i - \boldsymbol{u}_i^*$$

is the deviation from the optimal trajectory. $\boldsymbol{Q}$ and $\boldsymbol{R}$ are diagonal matrices where the elements define weights on the different states and input.

It is important to note that the $\boldsymbol{Q}$- and $\boldsymbol{R}$-matrix in this section is different from the matrices in Section 3. To avoid any kind of confusion the matrices for the LQ controller will be labelled $\boldsymbol{Q}_{LQR}$ and $\boldsymbol{R}_{LQR}$. Again, for simplicity $\boldsymbol{R}_{LQR}$ will be written in bold even though it is a scalar in this section. With the new notation the cost function is written as

$$\boldsymbol{J} = \sum_{i=0}^{\infty} \Delta\boldsymbol{x}_{i+1}^T \boldsymbol{Q}_{LQR}\boldsymbol{x}_{i+1} + \Delta\boldsymbol{u}_i^t \boldsymbol{R}_{LQR}\Delta\boldsymbol{u}_i \quad , \quad \boldsymbol{Q}_{LQR} \geq \boldsymbol{0}, \boldsymbol{R}_{LQR} \geq \boldsymbol{0} \tag{26}$$

This allows the helicopter to follow the trajectory until it reaches λ_f by use of the controller. At a given time, the reference is updated, allowing the controllers to track this reference actively rather than the open loop solution in section 3 where $\boldsymbol{u}_k$ is not modified according to the helicopters actual position. The cost function is minimized using the Riccati equation. In Matlab the Riccati equation for a discrete LQ problem for an infinite time horizon is (27) and (28)[6]

$$\boldsymbol{A}^T\boldsymbol{S}\boldsymbol{A} - \boldsymbol{S} - (\boldsymbol{A}^T\boldsymbol{S}\boldsymbol{B} + N)(\boldsymbol{B}^T\boldsymbol{S}\boldsymbol{B} + \boldsymbol{R}_{LQR})^{-1}(\boldsymbol{B}^T\boldsymbol{S}\boldsymbol{A} + N^T) + \boldsymbol{Q}_{LQR} = 0 \tag{27}$$

Where $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{Q}_{LQR}$ and $\boldsymbol{R}_{LQR}$ are defined, N is the time horizon and $\boldsymbol{S}$ is the solution to the Riccati equation. When N is omitted the default setting for Matlab is $N = 0$. When $\boldsymbol{S}$ is solved matlab solves for $\boldsymbol{K}$

$$\boldsymbol{K} = (\boldsymbol{B}^T\boldsymbol{S}\boldsymbol{B} + \boldsymbol{R}_{LQR})^{-1}(\boldsymbol{B}^T\boldsymbol{S}\boldsymbol{A} + N^T) \tag{28}$$

Another common form of the Riccati equation is (29)[7]

$$\boldsymbol{S} = \boldsymbol{Q}_{LQR} + \boldsymbol{A}^T\boldsymbol{S}(\boldsymbol{I} + \boldsymbol{B}\boldsymbol{R}_{LQR}^{-1}\boldsymbol{B}^T\boldsymbol{S})^{-1}\boldsymbol{A} \tag{29}$$

The problem was solved in Matlab using the *dlqr* function

```
    [K, S, e] = dlqr(A_d, B_d, Q_lqr, R_lqr)
```

The function returns optimal feedback gain $\boldsymbol{K}$, the solution to the Riccati equation $\boldsymbol{S}$ and the eigenvalues of the closed-loop system $\boldsymbol{e} = eig(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K})$.

[6]https://www.mathworks.com/help/control/ref/dlqr.html
[7]MPC note page 65 substitution P with S

See appendix B.2 for details of how the LQ controller was implemented in simulink. Furthermore, the control hierarchy of this part is displayed in Figure 8.

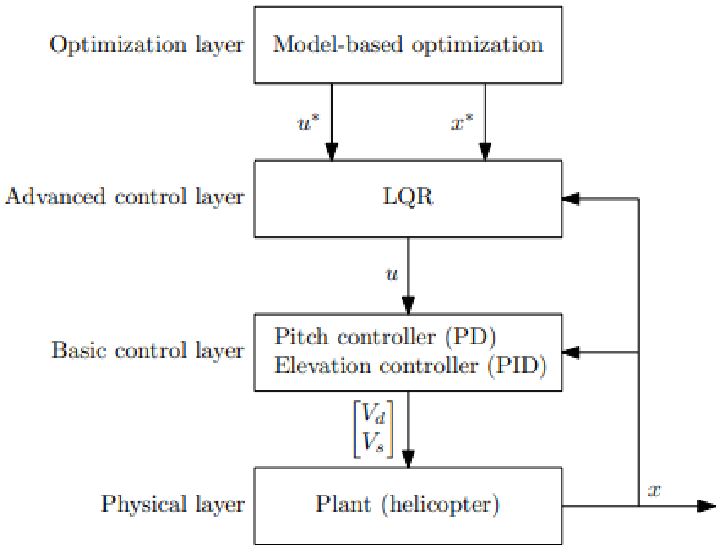


Figure 8: Control hierarchy with advanced control layer[8]

4.2 Part 2 - Simulation vs actual trajectory with feedback

When flying the helicopter, different values for the $\boldsymbol{Q}_{LQR}$- and $\boldsymbol{R}_{LQR}$-matrices were tested. The helicopter behaved relatively similar despite different weights on the states. In this case, travel is the most crucial state, therefore the starting point for tuning was set to: $\boldsymbol{Q}_{LQR} = diag([50\ 1\ 1\ 1])$ and $\boldsymbol{R}_{LQR} = 1$

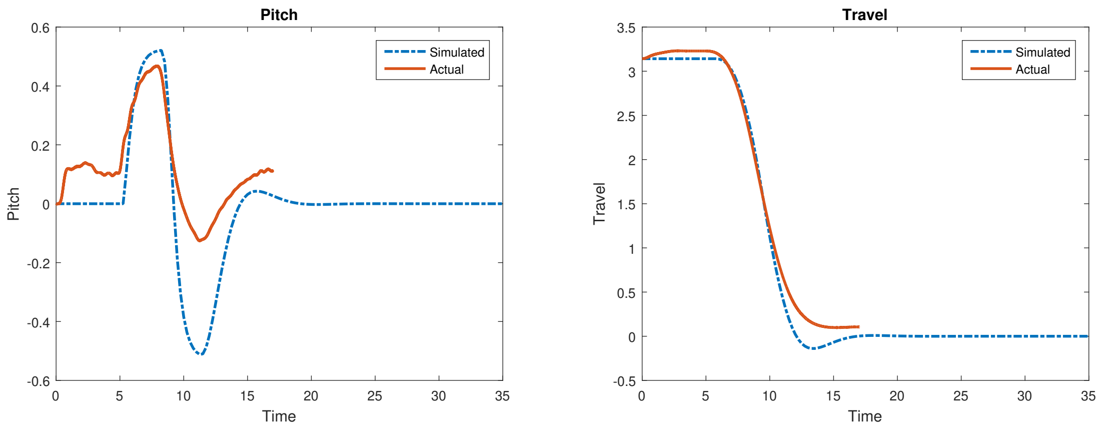


Figure 9: $\boldsymbol{Q}_{LQR} = diag([50\ 1\ 1\ 1]), \boldsymbol{R}_{LQR} = 1$

This resulted in a very accurate travel trajectory, as seen in Figure 9. However, pitch was not as accurate as in the simulation. A higher weight was set on pitch to see if it would result in an even more accurate trajectory:

[8]Copied from assignment text

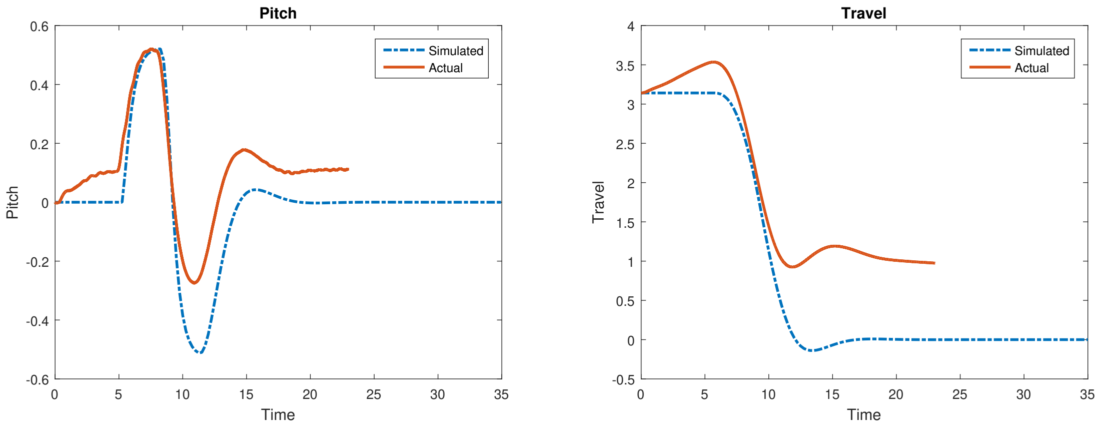


Figure 10: $\boldsymbol{Q}_{LQR} = diag([1\ 2\ 50\ 1]), \boldsymbol{R}_{LQR} = 1$

To no surprise the pitch trajectory is more similar to the simulated one, but this time travel is way off, as seen in Figure 10. As it is travel that is the most important state, it is more important to put weight on travel compared to pitch. The next test was to see how similar the helicopters flight could get to the simulated trajectory.

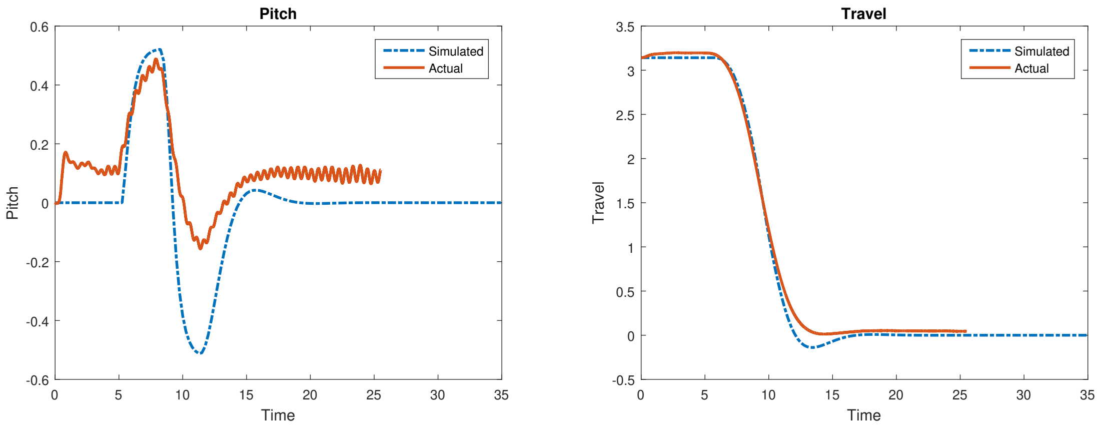


Figure 11: $\boldsymbol{Q}_{LQR} = diag([1000\ 1\ 1\ 1]), \boldsymbol{R}_{LQR} = 1$

As one can observe from Figure 11 the actual travel is near perfectly aligned with the simulation. However, the pitch is oscillating a lot, which usually cause problems on real life systems. From the plot (11) it looks like pitch is close to saturation. For a helicopter, an oscillating trip is not desirable. Even though the travel is very accurate, the parameters presented in Figure 9 might be a better choice of parameters. The final tests included putting restrictions on the input, and observe the impact it has on the trajectory. Lowering the values of R allows for more violent use of voltage.

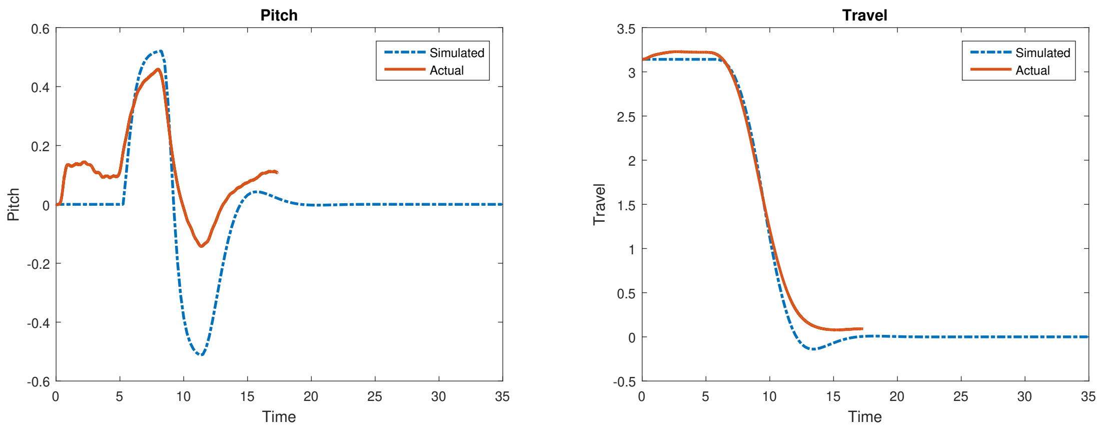


Figure 12: $\boldsymbol{Q}_{LQR} = diag([50\ 2\ 1\ 1]), \boldsymbol{R}_{LQR} = 0.1$

As shown in Figure 12, the helicopters actual trajectory is very similar to that in Figure 9. This shows that the weight on the R-matrix is not as important as long as the weight on travel the Q-matrix is of the magnitude it is. To double check R was set to 10.

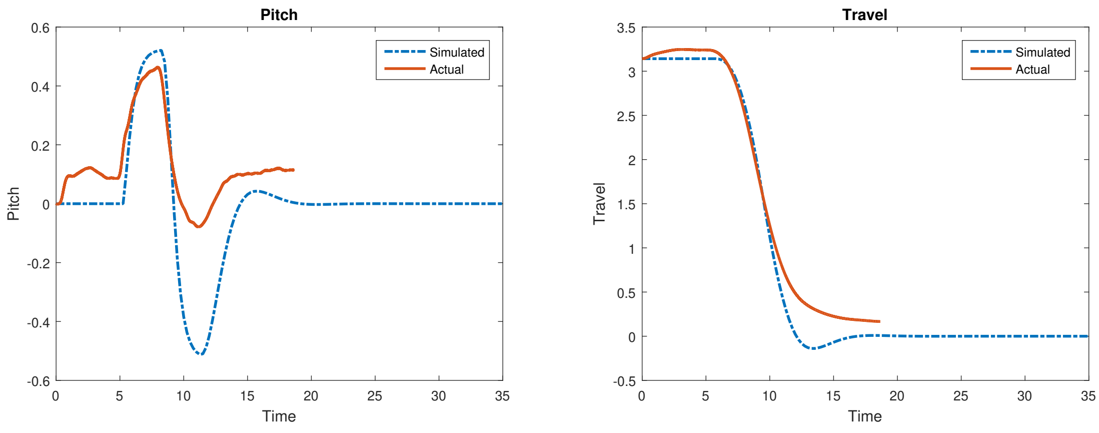


Figure 13: $\boldsymbol{Q}_{LQR} = diag([50\ 2\ 1\ 1]), \boldsymbol{R}_{LQR} = 10$

As shown in Figure 13, pitch and travel flattens out earlier than in the previous examples. However, as observed, relatively small changes to $\boldsymbol{R}_{LQR}$ makes little to no difference when there is such a high weight on travel.

4.3 Part 3 - MPC controller

If an MPC controller were to be implemented, The optimization problem would have to be solved over the time-horizon, N steps, for every time step based on the new reading of $\boldsymbol{x}_i$. This is, in other words, closed loop optimization. However, solving the optimization problem for each iteration using feedback would allow for more precise control, constantly modifying $[\boldsymbol{u}_i, ..., \boldsymbol{u}_{i+N}]$ in the event of an offset from the optimal trajectory.

In the event of a disturbance, a new route close to that of the original may be altered, paving way for more robust control (Merging Optimization and Control,pg 77 [2]). Furthermore, the MPC implementation would lead to a more accurate performance. In general, one may say that this feedback implementation of optimization leads to better and more robust control, but it comes at a cost.

Using an MPC requires more computing power, resulting in a higher power consumption. In a small device, this is a disadvantage. Also, since an MPC implementation requires $\Theta(N)$ running time for each time step (where N is the amount of steps for the time horizon), this may be unsuitable for systems with a small time step from $\boldsymbol{x}_k$ to $\boldsymbol{x}_{k+1}$.

To illustrate how MPC requires more computation, the algorithm found in Algorithm 7, Nonlinear MPC with state feedback in the book (Merging Optimization and Control pg. 72 [2]) is displayed below:

```
for t = 0, 1, 2, . . . do
    Get the current state x_t.
    Solve the optimization problem (4.41) on the
    prediction horizon from t to t + N with x_t as the initial condition.
    Apply the first control move u_t
    from the solution above.
end for
```

In comparison, in this project, the optimization problem is only solved once in advance and this sequence of values is used as a guideline for the helicopter to strictly follow, with or without control feedback.

4.4 Results and conclusion for the day

In this assignment, we have implemented a linear quadratic controller. Here, deviations from our desired trajectory, $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N]^T$ and $\boldsymbol{u} = [\boldsymbol{u}_0, \boldsymbol{u}_1, ..., \boldsymbol{u}_{N-1}]^T$ is compensated through the use of feedback to a controller. This allows us to much more accurately execute the maneuver compared to section 3. Furthermore, this approach is more robust against noise such as draft from an open window or a nudge because $\Delta\boldsymbol{x}$ would lead to a different and more suitable input value. By adding the reference 0 to travel rate both before - and after the maneuver, we accomplish what we want in a good way: Moving the helicopter from 0 to π

5 Exercise 4 - Optimal control of pitch/travel and elevation with and without feedback

5.1 Part 1 - State space for continuous time

The objective of this task is to calculate the optimal trajectory for two states, travel and elevation. With that in mind, the state space model has to be expanded to include elevation and elevation rate, resulting in a 6x1 state, and a 2x1 input

$$\boldsymbol{x} = \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \\ e \\ \dot{e} \end{bmatrix} \qquad \textit{and} \quad \boldsymbol{u} = \begin{bmatrix} p_c \\ e_c \end{bmatrix} \tag{30}$$

To make a new state space model on the form presented in equation (31), new values for the A and B matrices must be found.

$$\dot{\boldsymbol{x}} = \boldsymbol{A}_c\boldsymbol{x} + \boldsymbol{B}_c\boldsymbol{u} \tag{31}$$

Combining Equation 1 with the state space model results in the following state space model:

$$\begin{bmatrix} \dot{\lambda} \\ \dot{r} \\ \dot{p} \\ \ddot{p} \\ \dot{e} \\ \ddot{e} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3K_{ep} & -K_3K_{ed} \end{bmatrix} \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \\ e \\ \dot{e} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3K_{ep} \end{bmatrix} \begin{bmatrix} p_c \\ e_c \end{bmatrix} \tag{32}$$

5.2 Part 2 - State space for discrete time

For this part the goal was to discretize the system using the Euler method. The Euler method is

$$\frac{\boldsymbol{x}_{k+1} - \boldsymbol{x}_k}{\Delta t} = \boldsymbol{A}_c\boldsymbol{x}_k + \boldsymbol{B}_c\boldsymbol{u}_k \tag{33}$$

Solving the equation results in

$$\begin{aligned} \boldsymbol{x}_{k+1} &= (\boldsymbol{I} + \Delta t\boldsymbol{A}_c)\boldsymbol{x}_k + \Delta t\boldsymbol{B}_c\boldsymbol{u}_k \\ &= \boldsymbol{A}_d\boldsymbol{x}_k + \boldsymbol{B}_d\boldsymbol{u}_k \end{aligned} \tag{34}$$

With

$$\boldsymbol{A}_d = \boldsymbol{I} + \Delta t\boldsymbol{A}_c = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & -\Delta tK_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & -\Delta tK_1K_{pp} & 1-\Delta tK_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & -\Delta tK_3K_{ep} & 1-\Delta tK_3K_{ed} \end{bmatrix} \tag{35}$$

and

$$\boldsymbol{B}_d = \Delta t\boldsymbol{B}_c = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \Delta tK_1K_{pp} & 0 \\ 0 & 0 \\ 0 & \Delta tK_3K_{ep} \end{bmatrix} \tag{36}$$

Using $\Delta t = 0.25$ means

$$\boldsymbol{A}_d = \begin{bmatrix} 1 & 0.25 & 0 & 0 & 0 & 0 \\ 0 & 1 & -0.1415 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.25 & 0 & 0 \\ 0 & 0 & -0.81 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0.25 \\ 0 & 0 & 0 & 0 & -0.0625 & 0.75 \end{bmatrix} \tag{37}$$

has the eigenvalues $\lambda_{1,2,3,4,5,6} = 1, 1, 0.55, 0.55, 0.875, 0.875$ calculated by $eig(\boldsymbol{A}_d)$ in Matlab. Hence, the system is said to be marginally stable in discrete time. The discrete $\boldsymbol{B}$-matrix, with a time step $\Delta t = 0.25$, is

$$\boldsymbol{B}_d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0.81 & 0 \\ 0 & 0 \\ 0 & 0.0625 \end{bmatrix} \tag{38}$$

To check controlability, the ctrb function in Matlab was used:

```
sys = ctrb(A_d,B_d)
rank(sys)

ans =

     6
```

Thus the controlability matrix, $\boldsymbol{\mathcal{C}} = [\boldsymbol{B} \quad \boldsymbol{AB} \quad \boldsymbol{A}^2\boldsymbol{B} \quad \boldsymbol{A}^3\boldsymbol{B} \quad \boldsymbol{A}^4\boldsymbol{B} \quad \boldsymbol{A}^5\boldsymbol{B}]$ has full rank. This means the system is controllable, meaning every state can be controlled.

Clearly, the system is observable, since every state position can be read directly. From each position over time, the rate of the corresponding state can be calculated.

5.3 Part 3 - Optimization problem

Now an inequality constraint is implemented

$$e_k \geq \alpha(-\beta(\lambda_k - \lambda_f)^2) \quad \forall k \in \{1, \dots, N\} \tag{39}$$

This is a nonlinear constraint. The following cost function also needs to be minimized:

$$\phi = \sum_{i=0}^{N-1} (\lambda_{i+1} - \lambda_f)^2 + q_1 p_{ci}^2 + q_2 e_{ci}^2 \tag{40}$$

To handle this nonlinear constraint in Matlab, the optimization problem needs to be formulated as a standard QP problem. Because the cost function is quadratic, the linear term can be ignored. This results in:

$$\min_{\boldsymbol{z}} f(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^T\boldsymbol{G}\boldsymbol{z} \quad s.t \begin{cases} c_i(\boldsymbol{z}^*) = 0 & \in \mathcal{E} \\ c_i(\boldsymbol{z}^*) \geq 0 & \in \mathcal{I} \end{cases} \tag{41}$$

The $\boldsymbol{z}$-vector and $\boldsymbol{G}$-matrix will be on the same form as Equation 14 with mx increasing from 4 to 6 and mu increasing from 1 to 2. However, the constraint will be different. The nonlinear constraint is

$$c(\boldsymbol{x}_k) = \alpha(-\beta(\lambda_k - \lambda_f)^2) - e_k \leq 0 \tag{42}$$

This was implemented in Matlab as such:

```
c = alpha*exp(-beta*(x(1:6:6*N) - lambda_t).^2) - x(5:6:6*N)
```

Because the constraint is quadratic, the standard QP solver can not be used. Instead, the fmincon function was used, which is a SQP-type algorithm.

```
fun = @(z) 0.5z'*G*z;
opt = optimoptions('fmincon','Algorithm','sqp');
z0(1:mx) = x0;
tic;
[z,lambda] = fmincon(fun,z0,[],[],Aeq,Beq,vlb,vub,@gen_nonlin_constraints,opt);
t1 = toc;
```

The new states were then simulated for different values for q_1 and q_2, as presented in Figure 14, Figure 15 and Figure 16. As one can derive from the aforementioned figures, most states seem to flatten out with increasing values for q_1 and q_2

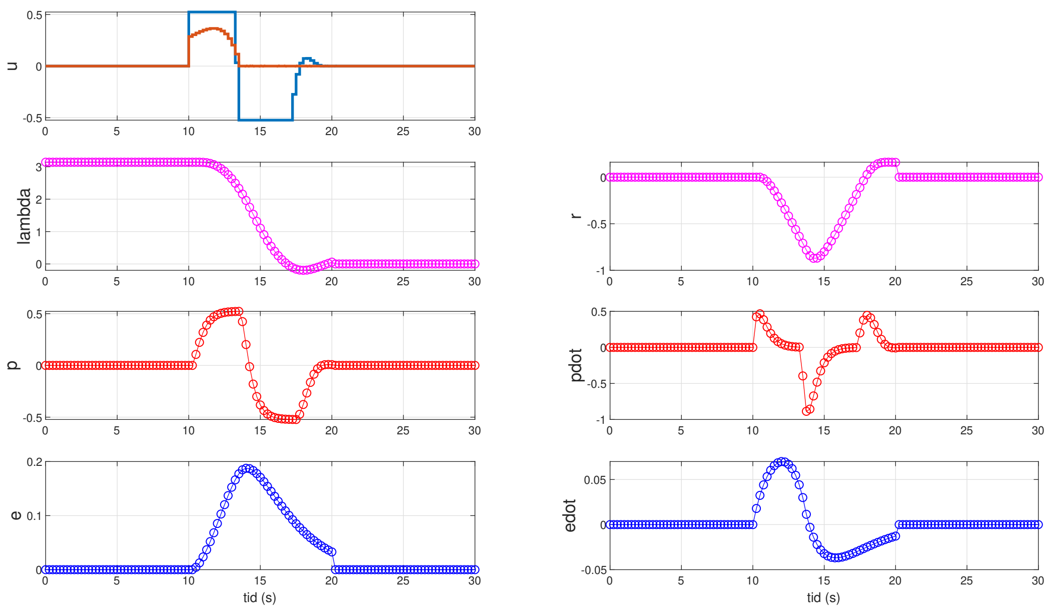


Figure 14: Simulates states with $q_1 = q_2 = 0.1$

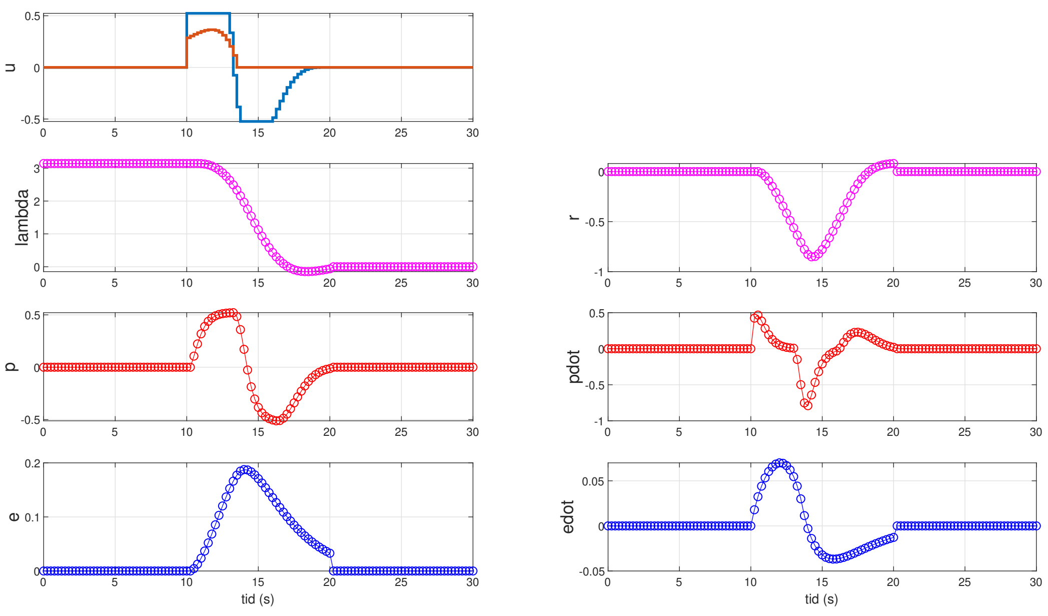


Figure 15: Simulates states with $q_1 = q_2 = 1$

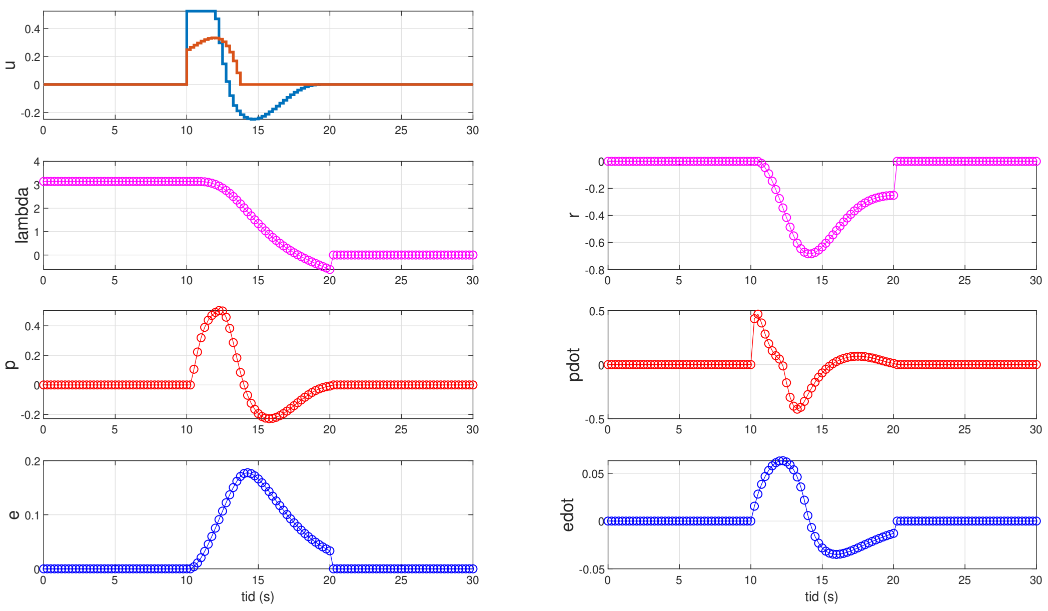


Figure 16: Simulates states with $q_1 = q_2 = 10$

5.3.1 Discussion

Note how in (14), p_{dot} changes very rapidly compared to (16) which has a $\boldsymbol{q}$ higher by a factor of 100. From this, we can argue that a very small q allows very sudden movements. This would, in theory, lead to quicker and more precise control. However, the inertia of the propellers, slight time delay and other factors will at some point make this impossible. Therefore, $\boldsymbol{q}$ needs to be tuned in such a way that he helicopter can in fact follow this trajectory. If there was an obstruction that the helicopter or robotic arm, for that matter, needed to avoid this could potentially lead to a crash. Also, very rapid movements in some systems may lead to failure or damage.

5.4 Part 4 - Simulation vs actual trajectory with and without feedback

Without feedback there is no correction in the actual states compared to the optimal trajectory. For elevation, drift is not visible in the same way as with travel. A certain voltage leads to a certain elevation. However, while not using feedback, the helicopter travel is allowed to drift, as an integral effect from an offset in pitch or an unforeseen initial velocity when the helicopter is supposed to stand still.

Also, the helicopter may follow a different trajectory than the one that is anticipated by the model. The reason for this is that the helicopter makes use of a given sequence of $\boldsymbol{u}$. In theory, this sequence should place the helicopter at the correct position (here π radians). In reality, imperfections in the model and disturbance causes the helicopter to end up in a location different from the goal. With feedback, these problems are eliminated to a satisfying degree. These arguments become clear when the plots presented in Figure 17 and Figure 18 are observed[9].

See appendix B.3 for the implementation in simulink with and without feedback.

[9]In these plots, an offset has been added to the pitch in order to make the helicopter stay still in travel-direction with reference $pitch = 0$.

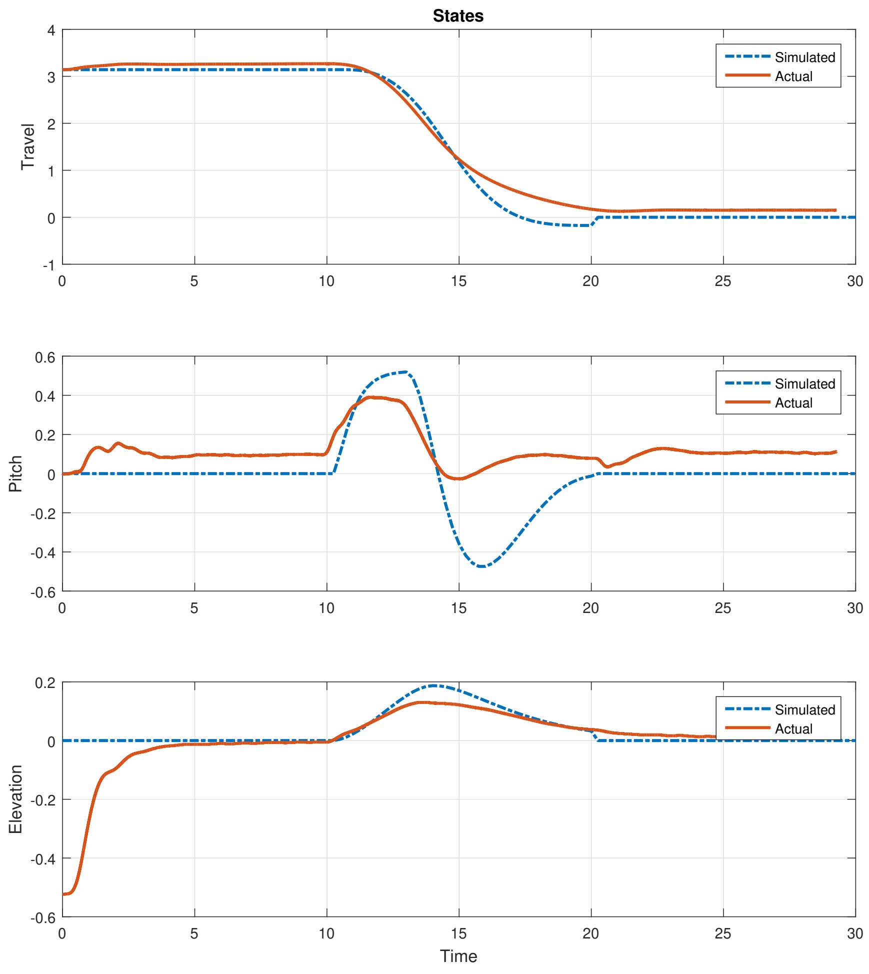


Figure 17: Simulated states and actual states with feedback

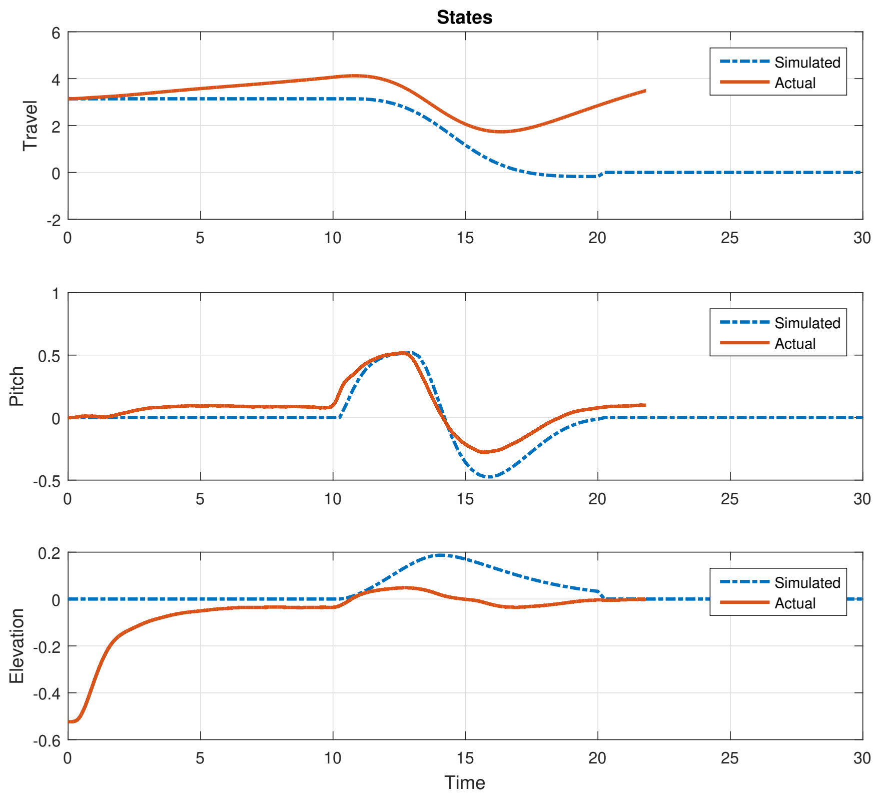


Figure 18: Simulated states and actual states without feedback

5.5 Part 5 - Decoupled states

In this model, elevation and elevation rate are decoupled from pitch, pitch rate, travel and travel rate. In reality, a change in pitch would affect elevation. In the case of a flight above a fictional obstacle, this assumption makes the helicopter fly at a lower altitude when pitching than the calculated route. One solution may be to take the trigonometric effect into account when modelling elevation and elevation rate. Another approach may be to set the trajectory higher, making the helicopter fly at a higher elevation for the entire duration of the flight. It is important to note that this is a result of the model itself, and not the optimization. However, by tuning the LQ controller the effect of the model can be negated by emphasising elevation

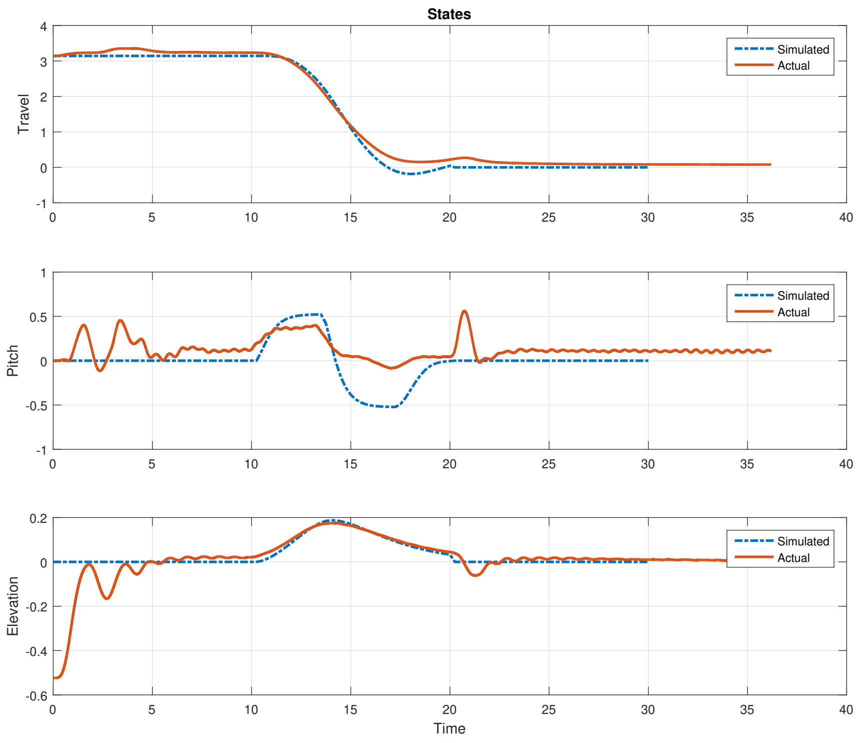


Figure 19: $\boldsymbol{Q}_{LQR} = diag([500\ 1\ 30\ 1\ 1000\ 1])$, $\boldsymbol{R}_{LQR} = diag([1\ 1])$

As seen in Figure 19 when emphasising travel and elevation, pitch is far from the optimal trajectory. it can also be observed that elevation is severely underdamped, which is not a desirable effect in a helicopter.

Geometrically, if the helicopter has a pitch angle θ, then the force working for elevation F_e and the net force from the propellers F has the relationship:

$$F_e = Fcos(\theta) \tag{43}$$

Hence, with an angle of $\frac{30\pi}{180}$, the force is 0.866 times that of the net force from the propellers. This, as well as the fact that the model is nonlinear, is where the models' imperfections become very visible in the trajectory. Therefore, at maximum allowed pitch, one can expect an elevation substantially lower than that of the simulated elevation unless error in elevation is highly penalized in the LQ controller.

5.6 Results and conclusion for the day

The most thought provoking aspect of section 5 is the impact the decoupling of states has on the elevation trajectory. From Figure 17, note that the actual elevation is in fact lower than the simulated trajectory. This is most likely a cause of the effects of decoupling. Note how the pitch is quite high when the elevation deviates the most from its simulated trajectory, while it follows its optimal trajectory quite nicely elsewhere.

6 Conclusion

In this assignment, we have used optimization to control a helicopter from point A to point B with both travel and elevation. In section 3, we implemented an open loop solution, which lead to a vulnerable control system unable to efficiently reach its destination. In section 4, we added feedback. Using an LQ controller, we managed to control the helicopter much more accurately. Finally, in section 5, we implemented a constraint in elevation. We discovered quickly that our assumptions for the model made the helicopter fly underneath the desired trajectory. This error in elevation was mostly eliminated by penalizing it quite strongly in the LQ controller.

All the exercises were done using open loop optimization. In theory, we would get better control using closed loop, although the precision we have obtained using LQ control with open loop optimization worked quite well.

A MATLAB Code

This section should contain your MATLAB code. DO NOT attach files posted online (that you didn't write). Note that the method used to input code below does not look as pretty when the lines are too long.

A.1 LQR.m

```matlab



Q_lqr = eye(6);
Q_lqr(1,1) = 1;      %Travel
Q_lqr(2,2) = 1;      %Travel rate
Q_lqr(3,3) = 1;      %Pitch
Q_lqr(4,4) = 1;      %Pitch rate
Q_lqr(5,5) = 1;      %Elevation
Q_lqr(6,6) = 1;      %Elevation rate
R_lqr = [1 0; 0 1]; %Input

[K, S, e] = dlqr(A_1, B_1, Q_lqr, R_lqr);
```

A.2 statesplt.m

```matlab

%Plot simulated states and actual states

figure(1)
subplot(3,1,1)
plot(t, x1, '-.', 'LineWidth', 2)
title('States')
ylabel('Travel')
hold on
plot(states.time, states.signals.values(:,1) + pi, '-', 'LineWidth', 2)
legend('Simulated', 'Actual')
grid('on')

subplot(3,1,2)
plot(t, x3, '-.', 'LineWidth', 2)
ylabel('Pitch')
hold on
plot(states.time, states.signals.values(:,3), '-', 'LineWidth', 2)
legend('Simulated', 'Actual')
grid('on')

subplot(3,1,3)
plot(t, x5, '-.', 'LineWidth', 2)
xlabel('Time')
ylabel('Elevation')
hold on
plot(states.time, states.signals.values(:,5), '-', 'LineWidth', 2)
legend('Simulated', 'Actual')
grid('on')
```

A.3 gen_nonlin_constraints.m

```matlab

```

```

function [c, c_eq] = gen_nonlin_constraints(x)
    N = 40;
    alpha = 0.2;
    beta = 20;
    lambda_t = 2*pi/3;

    c = zeros(N,1);

    c = alpha*exp(-beta*(x(1:6:6*N) - lambda_t).^2) - x(5:6:6*N);

    c_eq = [];
end
```

A.4 Optimization.m

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

%% Initialization and model definition
init06; % Change this to the init file corresponding to your helicopter

A_c = [ 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];
B_c = [ 0           0;
        0           0;
        0           0;
        K_1*K_pp    0;
        0           0;
        0           K_3*K_ep];

% Discrete time system model. x = [lambda r p p_dot e e_dot]'
Δ_t = 0.25; % sampling time
A_1 = [1     Δ_t      0                     0                   0 ...
                             0;
      0     1            -K_2*Δ_t          0                   0 ...
                                  0;
      0     0            1                    Δ_t              0 ...
                                  0;
      0     0            -K_1*K_pp*Δ_t    1-K_1*K_pd*Δ_t   0                     0;
      0     0            0                    0                   1 ...
                                  1*Δ_t;
      0     0            0                    0                   -K_3*K_ep*Δ_t ...
              1-K_3*K_ed*Δ_t];
B_1 = [ 0                   0;
        0                   0;
        0                   0;
        K_1*K_pp*Δ_t     0;
        0                   0;
        0                   K_3*K_ep*Δ_t];

% Number of states and inputs
mx = size(A_1,2);                        % Number of states (number of columns in A)
mu = size(B_1,2);                        % Number of inputs(number of columns in B)

% Initial values
x1_0 = pi;                               % Lambda
x2_0 = 0;                                % r (lambda_dot)
```

```matlab
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      = [-ul(1); 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); % 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 (objecitve 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
Q1(5,5) = 0;                            % Weight on state x5
Q1(6,6) = 0;                            % Weight on state x6
P1 = [10 0;
      0   10];                         % Weight on input
Q = gen_q(Q1,P1,N,M);                   % Generate Q, hint: gen_q
c = zeros((mx+1)*100,1);                % Generate c, this is the linear constant ...
    term in the QP





%% Generate system matrixes for linear model
Aeq = gen_aeq(A_1,B_1,N,mx,mu);
Beq_1 = A_1*x0;                         % Generate b
Beq = Aeq*z0;
Beq(1) = Beq_1(1);


%% Solve QP problem with linear model

fun = @(z) 0.5*z'*Q*z;
opt = optimoptions('fmincon','Algorithm','sqp');
z0(1:mx) =x0;
tic
[z,lambda] = fmincon(fun, z0, [],[], Aeq, Beq, vlb, vub, @gen_nonlin_constraints, ...
    opt);
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:mu:N*mx+M*mu)    z(N*mx+2:mu:N*mx+M*mu);
      z(N*mx+M*mu-1)            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)];
x5 = [x0(5); z(5:mx:N*mx)];             % State x5 from solution
x6 = [x0(6); z(6:mx:N*mx)];

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

u   = [zero_padding zero_padding; u; zero_padding 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 = [x5_0*unit_padding; x5; x5_0*unit_padding];
x6 = [zero_padding; x6; zero_padding];




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

figure(2)
subplot(421)
stairs(t,u,'LineWidth',2),grid
ylabel('u','FontSize',13)
subplot(423)
plot(t,x1,'m',t,x1,'mo'),grid
ylabel('lambda','FontSize',13)
subplot(424)
plot(t,x2,'m',t,x2','mo'),grid
ylabel('r','FontSize',13)
subplot(425)
plot(t,x3,'r',t,x3,'ro'),grid
ylabel('p','FontSize',13)
subplot(426)
plot(t,x4,'r',t,x4','ro'),grid
ylabel('pdot','FontSize',13)
subplot(427)
plot(t,x5,'b',t,x5,'bo'),grid
xlabel('tid (s)','FontSize',10), ylabel('e','FontSize',13)
subplot(428)
plot(t,x6,'b',t,x6,'bo'),grid
xlabel('tid (s)','FontSize',10),ylabel('edot','FontSize',13)
```

B Simulink Diagrams

B.1 A Simulink Diagram

Figure 20 shows the final system.

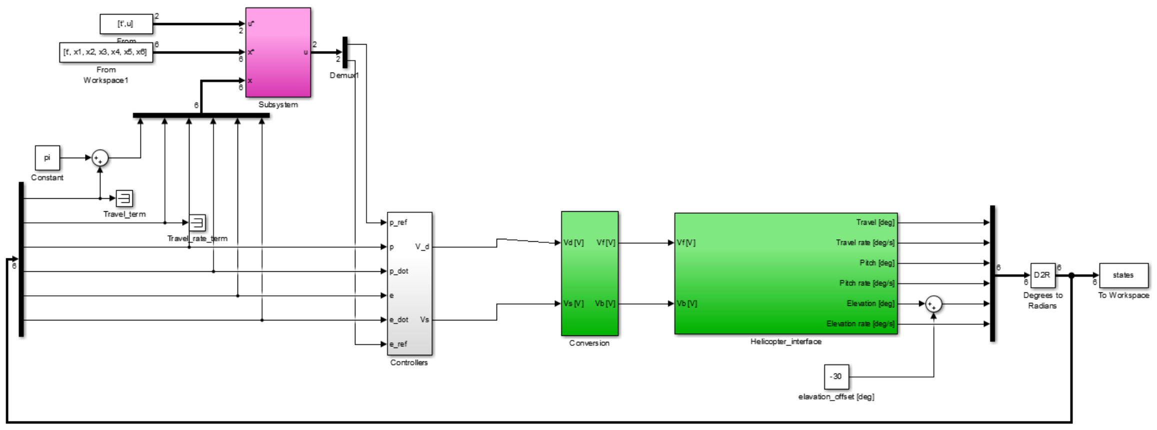


Figure 20: The final simulink diagram

B.2 LQ controller for part 4

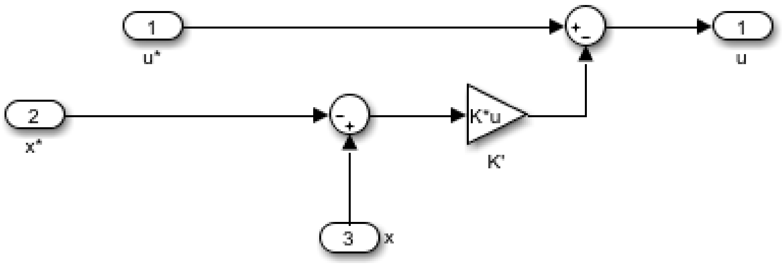


Figure 21: Subsystem for the LQ-controller

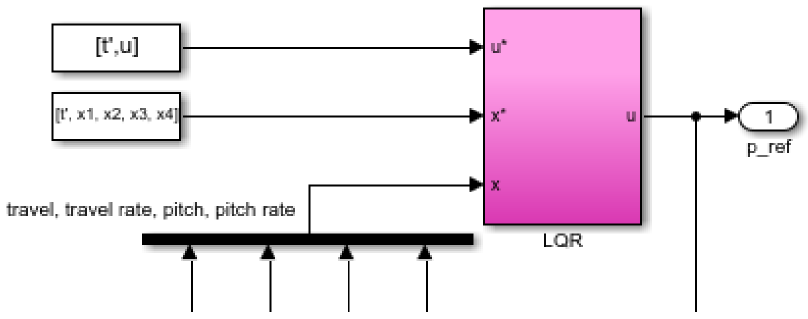
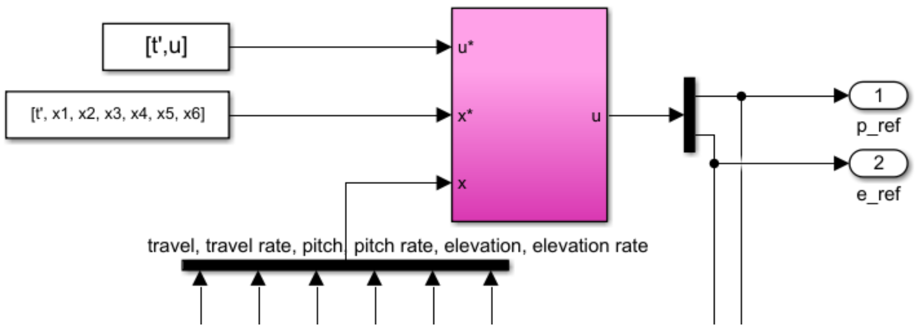


Figure 22: Overview of the LQ-controller

B.3 LQ controller for Part 5.4


[t',u]
[t', x1, x2, x3, x4, x5, x6]
travel, travel rate, pitch, pitch rate, elevation, elevation rate
u*
x*
x
u
1
p_ref
2
e_ref
Figure 23: Overview of the LQ-controller

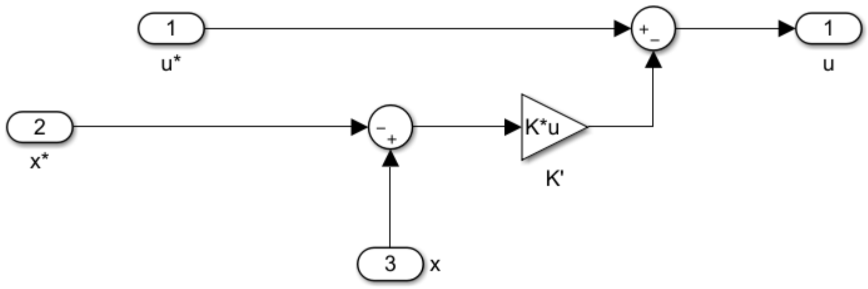


Figure 24: LQ-controller with feedback

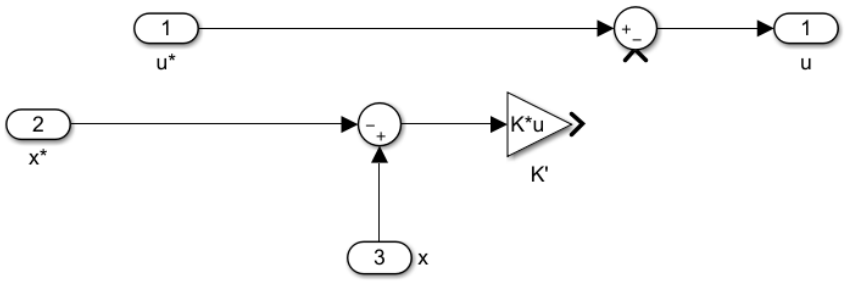


Figure 25: Without feedback

References

[1] Chi-Tsong Chen. *Linear System Theory and Design*. Oxford University Press, Incorporated, 2014.

[2] B Foss and T.A Heirung. *Merging Optimization and Control*. 2016.

[3] J. Nocedal and S. J. Wright. *Numerical Optimization*. second. Springer, 2006.