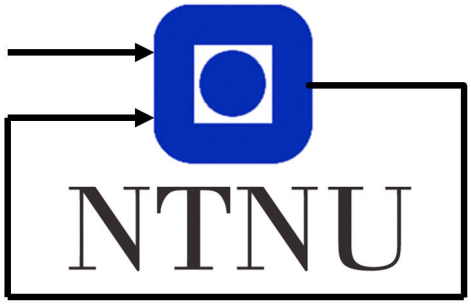
TTK4135 Helicopter Lab Report

Erik Furevik
Sindre Øversveen

Spring, 2021



Department of Engineering Cybernetics

Contents

1 10.2 - Optimal Control of Pitch/Travel without Feedback

1.1 The continuous model

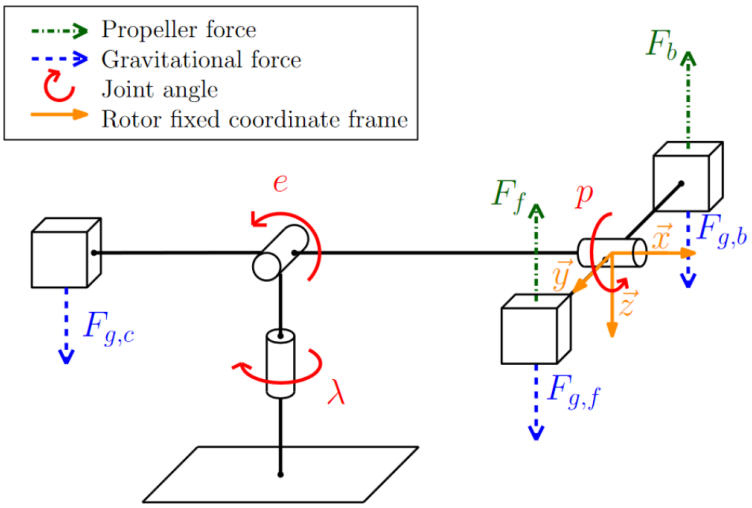


Figure 1: Helicopter sketch, from TTK4115 lab assignment text [1]. For illustration only: Pitch here is opposite from the actual system. Not to scale.

Figure 1 illustrates how the helicopter system is set up; what forces are in play, and the helicopter's range of movement. Our mathematical model of the system will be a set of differential equations of motion describing the forces shown in the figure. We use Newton's second law for rotation, $\tau = J\alpha$, to describe the acceleration around each axis e, λ and p.

We make several simplifications in deriving our model. For instance, it is assumed that the arms have no mass. This assumption is fairly reasonable in the context of rotation since most of the arm is significantly closer to the axis than the mass at the end, which means it contributes much less to the rotational inertia. The mass of the part of the arm furthest out may be described a part of the point mass. The rotational inertia around each axis is then given simply by $J = \sum mr^2$, where r and m are pairs of relevant lengths of arms and point masses. These are known constants.

Torque around an axis is given by $\tau = \sum Fr$, the sum of perpendicular forces of discance r from the axis. We assume the force from each propeller is proportional to applied voltage by a motor constant K_f. This is another simplification, as there is actually a lag caused by the acceleration time of the propellers. There are also aerodynamical properties to consider. For example, the helicopter will receive more thrust for a given voltage when

flying close to the table, because of *ground effect*. This effect is due to the table making it harder for the moved air to escape, resulting in decreased air flow and more resistance, thus more thrust for the propellers.

To formulate our model's equations of motion, we first set up Newton's second law around each axis. We then perform a linearization of the system, so we can utilize the toolbox of linear system theory and control. This involves assuming that $e \approx small$ and $p \approx small$. The first assumption is most reasonable, since the elevation when running will be usually only about ± 15 degrees to avoid crashing into the table. Pitch can take on a greater value, and will regularly be ± 30 degrees or more. With large pitch the force vector points in a different direction than straight down. This means that the lift decreases, and more force goes towards travel instead. But our model is essentially unaware of this connection due to the linearization. We can therefore expect a turn to large pitch to be followed by a drop in elevation.

The complete model derivation is given in chapter 8 in the lab-exercise handout [2], and will not be replicated here. We will simply note that the elevation dynamics are set up as a PID controller, the pitch dynamics as a PD controller, and the travel dynamics comes straight from the linearized physical model.

The complete set of equations of motion is

$$\begin{aligned}
\ddot{e} &= -K_3 K_{ed}\dot{e} - K_3 K_{ep} e + K_3 K_{ep} e_c \\
\ddot{p} &= -K_1 K_{pd}\dot{p} - K_1 K_{pp} p + K_1 K_{pp} p_c \\
\dot{\lambda} &= r \\
\dot{r} &= -K_2 p.
\end{aligned} \tag{1}$$

We will initially define our state vector as $x = [\lambda \ r \ p \ \dot{p}]^\top$; noting that we define our input $u = p_c$, we can write the system equations on state-space form $\dot{x} = A_c x + B_c u$ as

$$\begin{bmatrix} \dot{\lambda} \\ \dot{r} \\ \dot{p} \\ \ddot{p} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} \end{bmatrix} \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1 K_{pp} \end{bmatrix} p_c$$

From the selection of state variables, it is clear that we only model what is related to pitch and travel, and their rates. Elevation is not accounted for in the model, we simply assume that our PID controller is able to keep a steady elevation with a constant reference elevation. This implementation is not unreasonable, as the linearization of the model has removed any connection between elevation, and pitch/travel. This means that the optimization layer,

theoretically speaking, has no way of influencing elevation. Thus there is no reason to involve elevation is the optimization problem. The separation of elevation and pitch/travel is illustrated in figure 2.

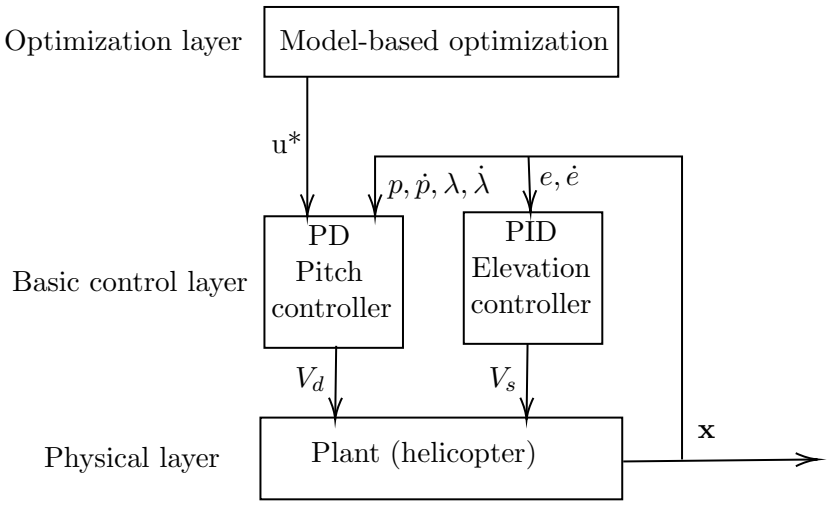


Figure 2: System control hierarchy, modified from lab exercise to show how elevation and pitch/travel are handled separately.

The model includes the physical helicopter, and how the PD pitch controller acts on it in the basic control layer. The model does not include the optimization layer in itself. Although the basic control layer is dependent on the optimal pitch reference, this reference is predetermined and the optimization layer is "out of the loop" once the helicopter starts running.

In the matrix equation, the first three rows come directly from physical relationships. Pitch rate is defined by pitch acceleration. Travel acceleration r is decided by pitch, since pitch causes force in travel direction. Finally travel rate is decided by travel acceleration.

Finally, our pitch acceleration is determined by the PD pitch controller through applied voltage V_d. This is the last row of the matrix equation. We see that there is a proportional K_{pp} gain on Δx and a dampening effect K_{pd} working on the derivative.

1.2 The discretized model

The definition of the derivative is

$$y'(a) = \lim_{h \to 0} \frac{y(a+h) - y(a)}{h}$$

The *forward Euler method* approximates this value by using a small $h = \Delta t$, and $a = t_0$ and $t_1 = t_0 + \Delta t$ at each timestep. For state space form we can

write

$$\begin{aligned} \dot{\boldsymbol{x_k}} &\approx \frac{\boldsymbol{x_{k+1}} - \boldsymbol{x_k}}{\Delta t} \approx A_c \boldsymbol{x_k} + B_c u_k \\ \implies \boldsymbol{x_{k+1}} &\approx \underbrace{(I + \Delta t A_c)}_{A} \boldsymbol{x_k} + \underbrace{\Delta t B_c}_{B} u \end{aligned} \tag{2}$$

The discretized matrices are thus

$$A = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & -\Delta t K_2 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & -\Delta t K_1 K_{pp} & 1 - \Delta t K_1 K_{pd} \end{bmatrix} \; and \; B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \Delta t K_1 K_{pp} \end{bmatrix}$$

1.3 The open loop optimization problem

The system should move from $x_0 = [\lambda_0 \; 0 \; 0 \; 0]^\top$ to $x_f = [\lambda_f \; 0 \; 0 \; 0]^\top$ within the given timeframe of 25 seconds, with elevation assumed constant. The objective function to be minimized is

$$\phi = \sum_{i=0}^{N-1} \left(\lambda_{i+1} - \lambda_f\right)^2 + q p_{ci}^2, \quad q \geq 0. \tag{3}$$

The optimization is subject to the the discrete state-space equation itself, and also the bounds

$$|p_k| \leq \frac{30\pi}{180}, \; \forall \; k. \tag{4}$$

The optimization will use a large vector $\boldsymbol{z}$ consisting of all optimization variables. These are N inputs, and N instances of each state. There is one state for $k \in \{1 \ldots N\}$ and one input for $k \in \{0 \ldots N-1\}$. They are displaced as such because x_0 is constant, while u_N has no effect. The vector has the form

$$z = \left[x_1^\top, \ldots, x_N^\top, u_0^\top, \ldots, u_{N-1}^\top\right]^\top.$$

The objective function should then be expressed in terms of this vector, which is done on the form

$$\min_z \frac{1}{2} z^\top G z + c^\top x, \text{ with constraints } \begin{cases} A \cdot z \leq b \\ A_{eq} \cdot z = b_{eq}, \\ z_{lb} \leq z \leq z_{ub}. \end{cases} \tag{5}$$

The G-matrix describes the weights on the quadratic optimization variables, so that the product $\frac{1}{2} z^\top G z$ equals the cost function (eq. 3). It is simply represented by a diagonal matrix with desired weights on the elements of the diagonal corresponding to the location of the variables in z. If Q is weight on the states x, then since only travel is represented in the cost

function, only the corresponding first element of Q is non-zero. We multiply G by 2, because eq. 5 multiplies the quadratic term by 0.5 compared to the formulation in eq. 3. Finally, as we have no linear term in the cost function, $c = 0$. Thus,

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \text{ and } G = 2 \begin{bmatrix} Q & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & q \end{bmatrix}$$

Moving on to the constraints – there are no inequality constraints in this optimization problem, and therefore $A = 0$ and $b = 0$. The state space model from eq. 2 will be implemented into the optimization problem as equality constraints, A_{eq} and b_{eq}. Writing out eq. 2 for each step, and making sure to put terms involving an optimization variable from z on the left hand side, we have

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

This is directly expressed in matrix form as

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

Finally, the bounds z_{lb}, z_{ub} are the limitations given by eq. 4. This is strictly speaking a bound on pitch only, but it is sensible to apply this to the input also, since the input is the pitch reference point. Also, there is a limit on voltage applied to the motors and physical limits on how much power the motors can apply. It is therefore good to put a bound on input, since the model doesn't otherwise take such limitations into account, and the result would be worse because of it. The upper bounds are thus

$$u_{ub} = p_{max} = \frac{30\pi}{180}, x_{ub} = \begin{bmatrix} \infty \\ \infty \\ p_{max} \\ \infty \end{bmatrix}, z_{ub} = \begin{bmatrix} x_{ub}^\top & \dots & x_{ub}^\top & u_{ub}^\top & \dots & u_{ub}^\top \end{bmatrix}^\top,$$

and similarly for the lower bounds z_{lb}. These are all the expressions necessary to formulate the problem on the standard form in eq. 5. Finally it is

noteworthy that reaching λ_f, which might be said to be the overall goal of the system, is not implemented directly as a constraint, as this goal is formulated through the objective function. More on this in section 1.5.

1.4 The weights of the optimization problem

The cost function in equation 3 takes into account travel, and input. On each of these two variables there is a weight. Clearly, a higher weight on one variable will make the value of that variable more important. And as the cost function should be minimized, placing a higher weight on one variable will mean that keeping this variable as low a possible will be of greater importance in the optimization.

It should also be noted that since there are only two variables to be weighted, it is sufficient to tune only one of them. That is to say, a reduction of the weight on one variable can always be achieved by putting more weight on the other. This is because we are only interested in the value that minimizes the cost function, not the actual value of the cost function itself. Thus, only the weights' sizes relative to each other is important. So for instance, multiplying the value of all weights by a scalar will change the value of the cost function proportionally, but will not affect the solution.

For this problem it should then be expected that putting a higher weight q on the input p_c^2, the optimal solution should be less willing to use large input, even at the cost of reaching desired travel more slowly. Similarly, using a lower input weight q should mean the optimal solution applies input more liberally to achieve a faster convergence. This theory was confirmed, as shown in figure 3.

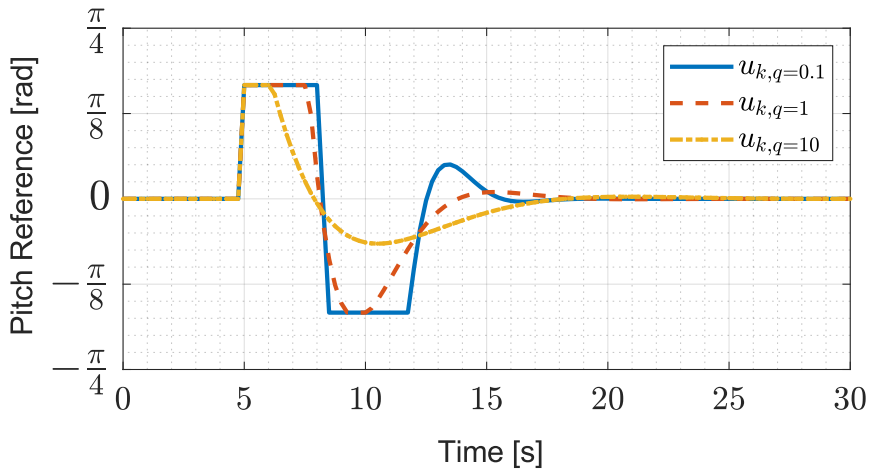


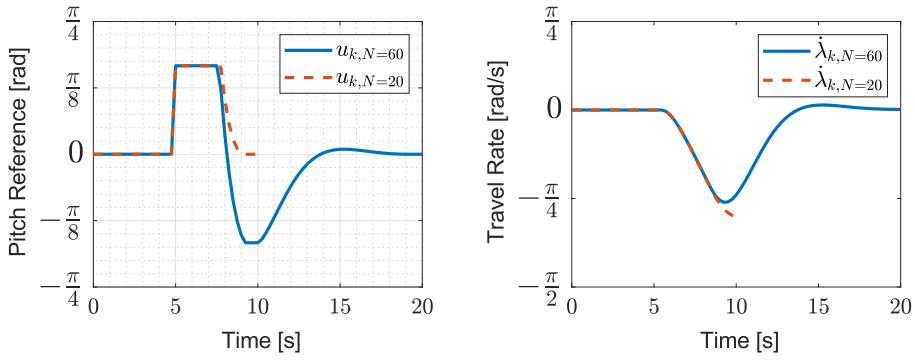
Figure 3: Optimal input with three different input weights q.

1.5 The objective function

The objective function is the sum of $(\lambda_i - \lambda_f)^2 + qp_c$ over all timesteps. The last term has already been discussed in the previous section, so this section will focus on the first term. We see that it looks at distance from the goal. This makes sense. However, when the helicopter reaches $\lambda = \lambda_f$, it will necessarily do so with a certain velocity. When it does, that term is zero, and so the simple solution to the cost function is to also set input to zero. Therefore, the helicopter might very well pass through the final point. So unless the optimal solution can stop the helicopter perfectly, which is unlikely since the model is not perfect, there will be a deviation.

Having the first term be quadratic means that the cost of large deviations in travel is very high, while the cost of small deviations is negligible. The optimal solution will therefore attempt to shorten the distance quickly, before shifting its focus to the pitch-term as tiny deviations in travel is neglected.

A potential problem with the formulation, appears if we shorten the time horizon for the optimization problem; if it is too short, the term $(\lambda_i - \lambda_f)^2$ makes it so the optimal solution will simply be to accelerate toward the target, with no plans for stopping – resulting in large overshoot. This is shown in figure 4. A "safer" problem formulation might have a constraint requiring zero velocity at the end of the time horizon. However, as long as the time horizon is made sufficiently large, this should not be an issue.



(a) Optimal input.

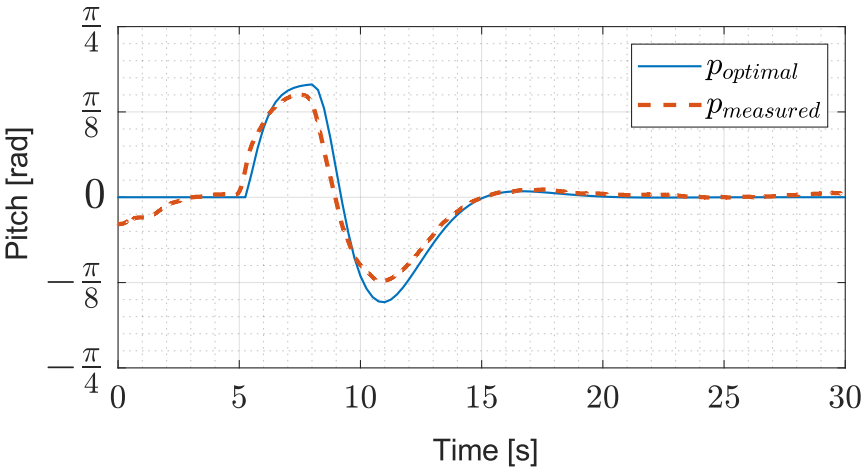
(b) Optimal travel trajectory.

Figure 4: Comparison of optimal u and λ for time horizons of 5 and 15 seconds. With short time horizon, the solution makes no attempt to slow down near the end.

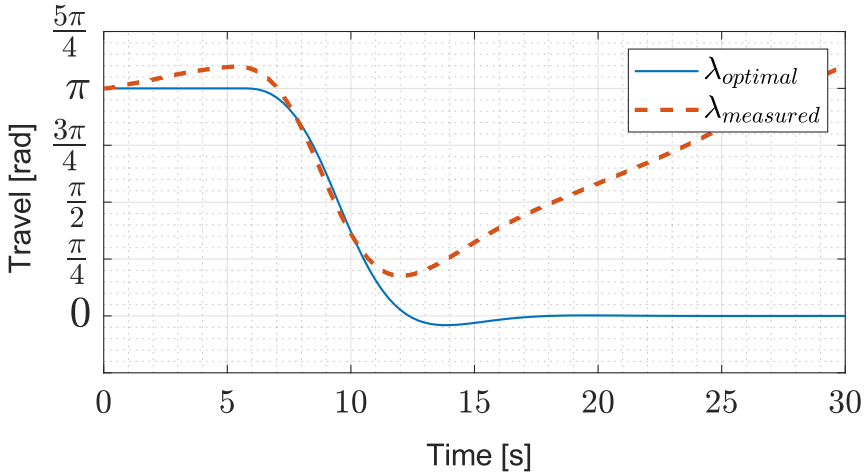
As a final comment, having the travel term squared might not be beneficial, depending on the application. For example, a human naturally thinks of distance linearly, and would usually consider having to walk 2km only 2x worse (linear) than having to walk 1km, not 4x worse (quadratic). A

quadratic response means the solution will try very hard to cover the first distance, but not so much the last part. This all depends on the application however, and what response is considered most beneficial.

1.6 Experimental results



(a) Measured pitch, and optimal pitch trajectory for comparison.



(b) Measured travel, and optimal travel trajectory for comparison.

Figure 5: Measured responses of travel and pitch, using $q = 1$. For comparison, the optimal open-loop trajectories have been added and labeled *optimal*. Note that optimal *state* is used for pitch, not to be confused with optimal *input*.

The response for $q = 1$ is shown in figure 5. In blue is the calculated optimal open-loop trajectory, for comparison. The pitch in 5a fits well with the calculated values from the preparation. This is to be expected, as the

pitch controller has feedback to ensure that the helicopter behaves as it should, and to handle any deviations from the pitch reference.

The measured travel in 5b fits fairly well with the optimal trajectory to begin with, and almost reaches the desired target. From there on however, it quickly diverges. The reason for this deviation is our imperfect system model – there are many assumptions and simplifications made in developing the model, as discussed in section 1.1. And since there is no feedback on travel, there is no way for the controller to correct for these deviations.

As for why the deviations for travel happen specifically in the positive direction, is hard to say, but here is one hypothesis: The optimal solution for any q generally first accelerates by pitching, then brakes by pitching oppositely. But, the acceleration is generally more agressive than the braking. I.e. it has higher amplitude in pitch and lasts for less time. See figure 3. The solution clearly believes that each stage provides equal change of momentum, to reach zero velocity at the end. However, it seems that the long and steady braking in reality gives more change of momentum than the shorter and more agressive acceleration. This difference in effect could in turn be partly explained by the lag also discussed in section 1.1. If there is a timelag in accelerating the propellers, of a constant time length, this would disproportionately affect the acceleration stage since it is shorter. Additionally, the system model relies on the assumption that $p = 0$ as part of its linearization – large deviations from the linearization point could contribute to the deviations in travel. Lastly, one must consider the possibility that the table upon which the helicopter is mounted, is not perfectly level. This could result in a natural rotation along the travel axis, due to gravitational pull – granted the table is sufficiently skewed.

However, the biggest weakness of the system is of course the lack of feedback. This makes it impossible for the optimization/control layer to know whether the helicopter is behaving as it should, and correct potential deviations.

It would be possible to improve this open-loop reponse even without using feedback. A more advanced, possibly nonlinear, model could be constructed, taking more factors into account to more accurately describe the real system. This in turn would give a better and more realistic optimization, and less deviation.

But this would come at the cost of a more challenging optimization problem to both formulate and solve, and even then there would still be errors. The most effective way to improve the result is therefore to introduce a feedback controller to handle any deviations from the planned trajectory. This is especially a feasible strategy since we already have very good state measurements available for feedback, we just have to implement the controller.

1.7 MATLAB and Simulink

Figure 6 shows the changes made to the simulink template. The optimal input is used as p_{ref} using a "From Workspace" block. A "To File" block is used to store the measurements. A "Selector" block is used to select only the relevant signals. An offset to pitch of -7 degrees was found experimentally to counteract a consistent drift in travel observed without input.

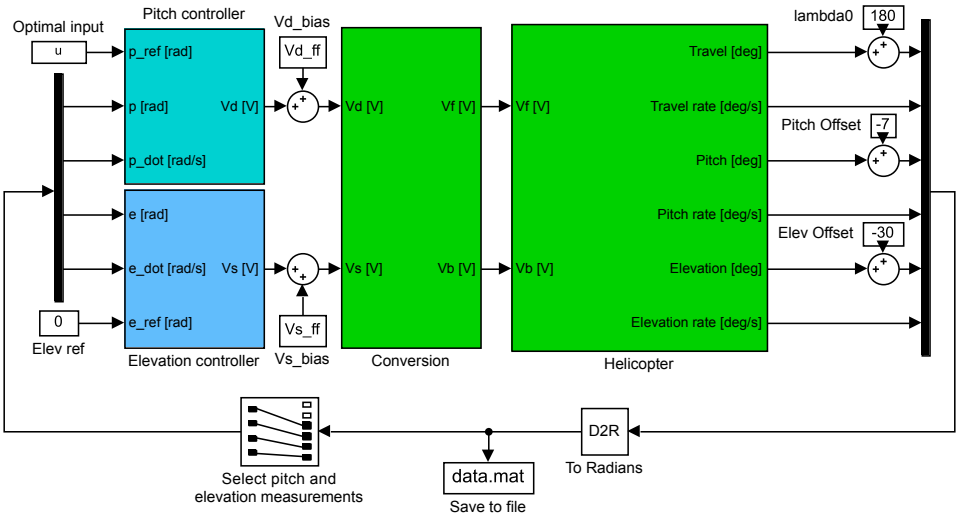


Figure 6: Full simulink diagram. Based on provided template.

Listing 1: Task 2 Matlab code.

```
% TTK4135 - Helicopter lab
% Template spring 2018, Andreas L. Fl ten

%% Initialization
init01                          % Load helicopter parameters

% Continuous model
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];

% Problem constants
N  = 100;                       % Time horizon
nx = width(Ac);                 % Number of states
nu = width(Bc);                 % Number of inputs
```

```
Nx = N*nx;                  % Total state variables
Nu = N*nu;                  % Total input variables
Nz = N*nx+N*nu;             % Total n. of optim. vars

% Discrete model by forward euler
Dt = 0.25;
Ad = Dt*Ac + eye(nx);
Bd = Dt*Bc;

% Initialization
lambda0 = pi;
x0 = [lambda0; 0; 0; 0];

%% Generate bounds
xl = -Inf*ones(nx,1);    % Lower bound (assume none)
xu = Inf*ones(nx,1);     % Upper bound (assume none)

pmax = 30*pi/180;           % Max allowed pitch either way
ul         = -pmax;             % Min pitch ref
uu         =  pmax;             % Max pitch ref
xl(3) = -pmax;              % Min pitch
xu(3) =  pmax;              % Max pitch

[zlb,zub] = gen_constraints(N,N,xl,xu,ul,uu);
zlb(Nz)  = 0;               % We want last input zero
zub(Nz)  = 0;               % We want last input zero

%% Objective function weights
Q = zeros(nx);
Q(1,1) = 1;                     % Travel term
q = 1;                          % Weight on input
G = 2*gen_q(Q,q,N,N);           % Generate G matrix
c = zeros(Nz,1);                % No linear term in
    function

%% Equality constraints
Aeq = gen_aeq(Ad,Bd,N,nx,nu);
beq = zeros(size(Aeq,1),1);
beq(1:nx) = Ad*x0;

%% Solve problem using quadprog
opt = optimset('Display','off', 'Diagnostics','off', '
    LargeScale','off', 'Algorithm', 'interior-point-
    convex');
```

```
60 z = quadprog(G,c,[],[],Aeq,beq,zlb,zub,[],opt);
61
62 %% Extract control inputs and states
63 padLength = 5/Dt;
64 zeroPad = zeros(padLength,1);
65 iPad = ones(padLength,1);
66
67 % v = [5s padding;   optimal solution;  padding]
68 u   = [zeroPad;      z(Nx+1:Nz); z(Nz); zeroPad];
69 x1  = [lambda0*iPad; x0(1);z(1:nx:Nx);  zeroPad];
70 x2  = [zeroPad;      x0(2);z(2:nx:Nx);  zeroPad];
71 x3  = [zeroPad;      x0(3);z(3:nx:Nx);  zeroPad];
72 x4  = [zeroPad;      x0(4);z(4:nx:Nx);  zeroPad];
73
74 t = 0:Dt:Dt*(length(u)-1);
75 tu = [t' u];
76
77 %% Plotting
78 figure(2)
79 subplot(511),stairs(t,u),      grid,ylabel('u')
80 subplot(512),plot(t,x1,'-mo'),grid,ylabel('lambda')
81 subplot(513),plot(t,x2,'-mo'),grid,ylabel('r')
82 subplot(514),plot(t,x3,'-mo'),grid,ylabel('p')
83 subplot(515),plot(t,x4,'-mo'),grid,ylabel('pdot')
84 xlabel('tid (s)')
```

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

2.1 LQ controller

A *Linear Quadratic Controller* is so named because it takes in a Linear system (section 1.2), and considers a Quadratic cost function on the form

$$\min f(u) = \sum_{k=0}^{\infty} \frac{1}{2}(x_{k+1}^T Q_{lqr} x_{k+1} + u_k^T R_{lqr} u_k).$$

It then finds the Controller; the feedback gain K that minimizes the cost function using the feedback law

$$\begin{aligned} &u_k = -Kx_k, \text{ or in our case, through change of variables;} \\ &u_k = u_k^* - K(x_k - x_k^*), \end{aligned} \tag{6}$$

where optimal input and state are marked by *.

Solving the equation requires solving the *Ricatti equation*. If the equation above has a finite horizon, it is solved by induction backwards in time. In this case, there may be a different K for each timestep – but since our particular formulation has an infinite horizon, the solution will instead be a constant matrix. To find the solution to the Discrete-time Algebraic Ricatti Equation (DARE),

$$P = A^\top PA - A^\top PB(R + B^\top PB)^{-1}B^\top PA + Q,$$

an iterative process is used to find the stabilizing solution. The gain matrix K can be shown to be given by

$$K = (R + B^\top PB)^{-1}B^\top PA.$$

There are alternative methods of designing the feedback gain, such as designing K directly, or using pole placement. One can rank these in terms of the level of abstraction;

1. Select feedback gain K directly.
2. Design K through selecting poles of closed-loop system.
3. Design K by LQR.

Generally speaking, a higher number on this list means that the system is easier to design in an intuitive sense, i.e. the engineer requires less specific understanding of the system to reach an acceptable feedback, and it is easier to tune a larger system due to the abstraction. One might say that the inner workings of the system are more hidden from view. A lower number on the

list can be considered more low-level design and means the engineer requires a deeper and better understanding of the methods. There would, however, be a more direct connection between tuning and resulting behaviour, and perhaps the possibility of achieving a better end result.

With its high level of abstraction, an LQR is quite simple to use. It is however more challenging to see how the tuning of various parameters affect the behaviour. The parameters to be tuned are matrices Q and R. These are almost always made diagonal for simplicity - if not, the high number of tuning variables will make the task unfeasible. To more clearly illustrate their importance, we might expand the cost function expression;

$$\begin{aligned}\min f(u) \sum_{k=0}^{\infty} & Q_{11}(\lambda_{k+1} - \lambda^*_{k+1})^2 + Q_{22}(r_{k+1} - r^*_{k+1})^2 + \\ & Q_{33}(p_{k+1} - p^*_{k+1})^2 + Q_{44}(\dot{p}_{k+1} - \dot{p}^*_{k+1}) + R(u_k - u_k)^2, \end{aligned} \tag{7}$$

where Q_{ii} is the i'th element of the diagonal of Q. Since there is only one input $u = p_c$, R is a 1x1 matrix (= scalar). First, like the weights of the objective function in the previous section, reducing R by a factor is equivalent to increasing Q by the same factor, and vice versa. For this reason, we can start by simply setting $R = 1$, and focusing our attention on Q only.

The elements of the diagonal of Q correspond to each state as shown in eq. 7. A higher weight on a state, which in this case are the deviations from optimal trajectory, will make such a deviation more costly in the cost function. Ideally, we would therefore simply consider the relative importance we place on each state, and placing the weights accordingly. Immediately it should be considered natural to put the most weight on travel, since changing this state is the overall goal of the optimization. To further speak on the relative value of the other states, we should repeat how the states generally affect each other,

$$V_d \rightarrow \ddot{p} \rightarrow \dot{p} \rightarrow p \rightarrow \dot{r} \rightarrow r \rightarrow \lambda.$$

An arrow $A \rightarrow B$ in this context means that state A decides the change in state B while state B affects not state A. We might call it a causal chain. This interconnectivity means there is no simple way to improve the response of only one or two states. The weights of Q must therefore all be weighed relative to each other.

The model is not an accurate description of the system, as seen in the previous exercise (figure 5). For each step in the chain, a certain amount of error is introduced, due to the inaccuracy of the model compared to the real system. This error propagates down the chain.

Therefore, correcting a state that is later in this chain is more likely to have a desired impact on the final state travel, due to there being less potential error introduced between them. Another way to look at it is that the "optimal trajectory" calculated for states closer to travel, such as travel rate, are more likely to be the true optimal trajectories. Or in yet another formulation, we can view the early states as being simply a means to an end (travel), therefore their values are less important as long as they have the desired impact on travel. The conclusion anyhow is that weights in Q corresponding to states that are closer to travel should be higher than those corresponding to less directly related states.

In practice there is always a certain amount of trial and error. The final weights used were

$$Q = \begin{bmatrix} 6 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0.1 \end{bmatrix} \text{ and } R = 1 \implies K = \begin{bmatrix} -1.9106 \\ -4.9996 \\ 2.1683 \\ 0.5284 \end{bmatrix}.$$

Using higher weights on pitch and pitch rate quickly lead to undesireable oscillations in pitch. And as established above, there is no real reason to put too much importance on these states anyhow. The weights follow the hierarchy of importance discussed above, i.e. the weights roughly corresponds to how directly the state is related to travel.

This controller could easily be tuned more agressively, but it should be remembered that its purpose is only to remove the deviations found in the previous exercise. The optimization layer is still responsible for the larger part of the operation, while the LQR works a supporting role.

2.2 Model Predictive Control

Using MPC involves calculating the open-loop optimal solution from each timestep while the helicopter is running, based on current measurements. This optimalization is often finite-horizon and/or with input blocking to increase speed, since it must be completed within a timestep. MPC would be realized in this case by solving the open-loop optimization problem in the previous task for each timestep, each time using updated initial conditions.

For the LQ controller as implemented in this task, we calculate an optimal trajectory and use LQR to follow it as closely as possible. The advantage is that solving the optimization problem and finding the optimal feedback gain can be done in advance, thus making the final controller very lightweight.

MPC instead finds a whole new trajectory each step. This means that it will be better able to handle errors and deviations, since it can always find a new trajectory. Given a deviation from optimal trajectory, the LQ controller will always try to return to it, which might be unnecessary. MPC will instead simply find the best way forward, given the current situation. For this reason, MPC can be considered more dynamic and flexible. However, this of course comes at the cost of being more computationally demanding.

It should also be noted that MPC will take constraints, even nonlinear, into consideration when running, and always try to avoid them. The LQ controller tries to follow a precalculated trajectory. This trajectory takes constraints into account, but the regulator responsible for following it does not. This means that, especially in the case of nonlinear constraints and with some deviation, the LQR can move the helicopter to violate constraints.

Figure 7 shows how the control hierarchy would look if we used MPC instead of LQ controller.

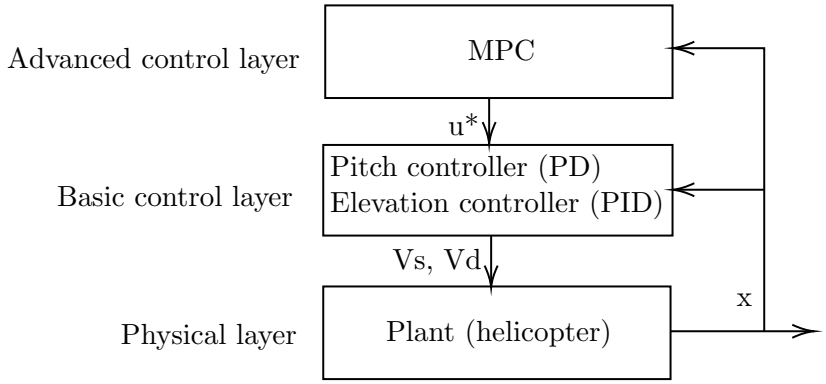
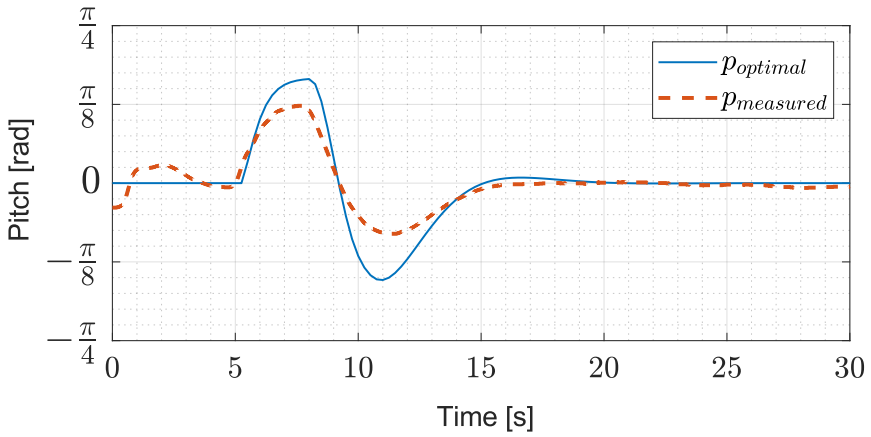


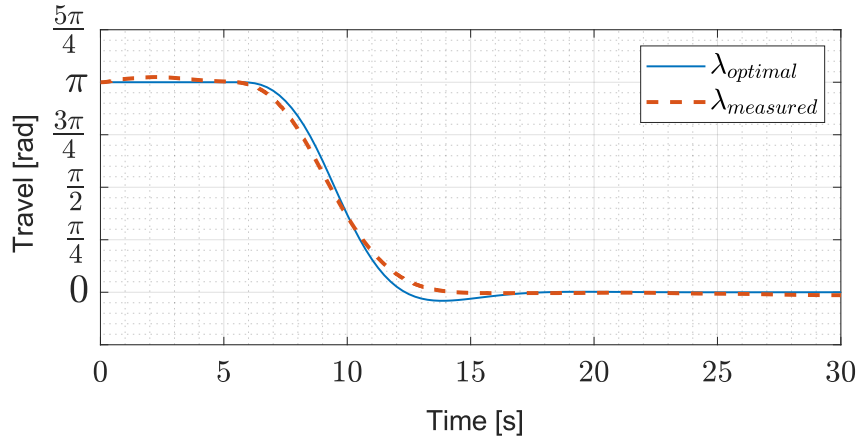
Figure 7: Control hierarchy if using MPC. Based on Figure 8 in the lab exercise.

2.3 Experimental results

Figure 8 shows the experimental results with LQ controller, compared to the optimal trajectory. The response is much better than without feedback (figure 5). Pitch does not follow the optimal trajectory perfectly, but this is to be expected as the chosen weights Q put little emphasis on this state. The important result is that the final desired state x_f is now reached, with a smooth response and very little deviation in travel.



(a) Measured vs optimal pitch.



(b) Measured vs optimal travel.

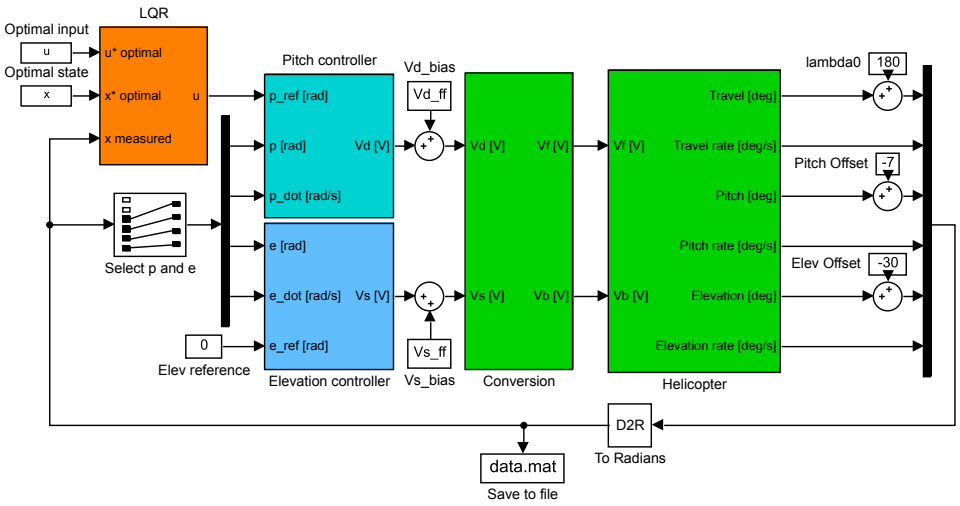
Figure 8: The measured values for pitch and travel, compared to the optimal open-loop trajectories.

2.4 MATLAB and Simulink

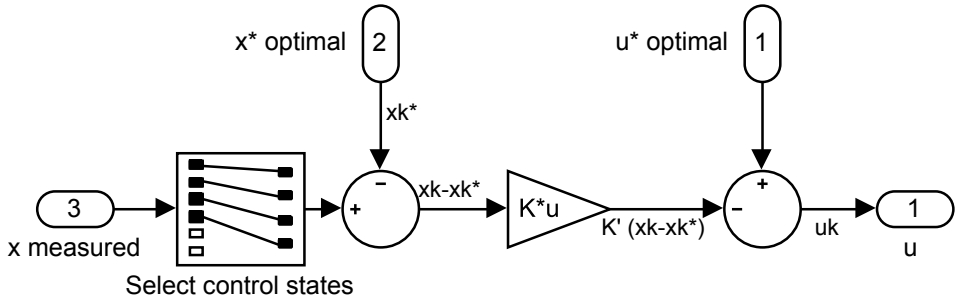
The full simulink diagram is shown in figure 9a. The only difference from the previous lab exercise is the addition of the LQR. This controller requires the optimal states $x*$ from the open-loop optimalization, in addition to the optimal input $u*$. These are added using "From Workspace" blocks.

The LQR itself is placed into a subsystem to simplify the design. The contents of this subsystem is shown in figure 9b. Since only travel and pitch are included in the optimalization, these are the states that should be controlled by the LQR. A selector block is therefore used to select travel and pitch states (including their rates). The subsystem then calculates input according to the feedback rule, eq. 6.

This system uses the same optimal open-loop solution as the previous exercise. For this reason the matlab code simply runs the previous exercise, and then finds the optimal LQ controller gain K given the chosen weights R_{lqr} and Q_{lqr}.



(a) Full simulink diagram.



(b) Simulink LQR subsystem.

Figure 9: The simulink system. The LQR calculation is made a subsystem to make the system easier to understand.

Listing 2: Task 3 Matlab code.

```
% Solve optimization problem
task2 % runs code from previous task
% Find optimal LQ controller
Qlqr = diag([6 2 0.2 0.1]);
Rlqr = 1;
Klqr = dlqr(Ad,Bd,Qlqr,Rlqr);
% For simulink:
x = [t' x1 x2 x3 x4];
```

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

3.1 The continuous model

As we now wish to also control elevation, we return to the equations of motion 1, and expand our state-space to include the elevation dynamics. We redefine our state as $x = [\lambda\ r\ p\ \dot{p}\ e\ \dot{e}]^\top$, and note that by defining our input as $u = [p_c\ e_c]^\top$, we can write the system equations on continuous state-space form $\dot{x} = A_c x + B_c u$ as

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

3.2 The discretized model

As before, the forward Euler method yields the discrete state-space form

$$\boldsymbol{x_{k+1}} \approx \underbrace{(I + \Delta t A_c)}_{A}\boldsymbol{x_k} + \underbrace{\Delta t B_c}_{B} u$$

with the discretized matrices now being

$$A = \begin{bmatrix}1 & \Delta t & 0 & 0 & 0 & 0\\ 0 & 1 & -\Delta t K_2 & 0 & 0 & 0\\ 0 & 0 & 1 & 1 & 0 & 0\\ 0 & 0 & -\Delta t K_1 K_{pp} & 1-\Delta t K_1 K_{pd} & 0 & 0\\ 0 & 0 & 0 & 0 & 1 & \Delta t\\ 0 & 0 & 0 & 0 & -\Delta t K_3 K_{ep} & 1-\Delta t K_3 K_{ed}\end{bmatrix}$$

and $B = \begin{bmatrix}0 & 0 & 0 & \Delta t K_1 K_{pp} & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & \Delta t K_3 K_{ep}\end{bmatrix}^\top$

Note than in this part, an additional nonlinear constraint was put upon elevation, in the form of

$$e_k \geq \alpha e^{\left(-\beta(\lambda_k - \lambda_t)^2\right)} \quad \forall k \in \{1, \ldots, N\} \tag{8}$$

with $\alpha = 0.2$, $\beta = 20$ and $\lambda_t = \frac{2\pi}{3}$.

3.3 Experimental results

After some tuning, an acceptable LQR was reached, using diagonal weighting matrices with values $Q_{diagonal} = (10\ 2\ 0.2\ 0.1\ 10\ 5)$ and $R_{diagonal} = (1\ 1)$, resulting in the feedback gain

$$K = \begin{bmatrix} -2.4130 & -5.8336 & 2.4417 & 0.5814 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.3569 & 0.4067 \end{bmatrix}$$

Figure 10 shows the results. Results in pitch and travel, both the optimal and measured trajectories, are very similar to the previous exercise. It makes sense that the added elevation constraint should not much affect these states, due to our decoupled model, discussed in the next section.

The interesting state is elevation. First, we should note that not even the optimal trajectory seems to fulfill the constraints. This is due to the level of discretization of the optimalization. For the purpose of this discussion, optimal elevation (blue) for each timestep is marked with a circle. We see that these do fulfil the constraint. This proves that the optimalization does not think continuously, i.e. it uses small "jumps". We should therefore be aware that, even if the optimal trajectory was followed perfectly, the helicopter trajectory would still violate the constraints.

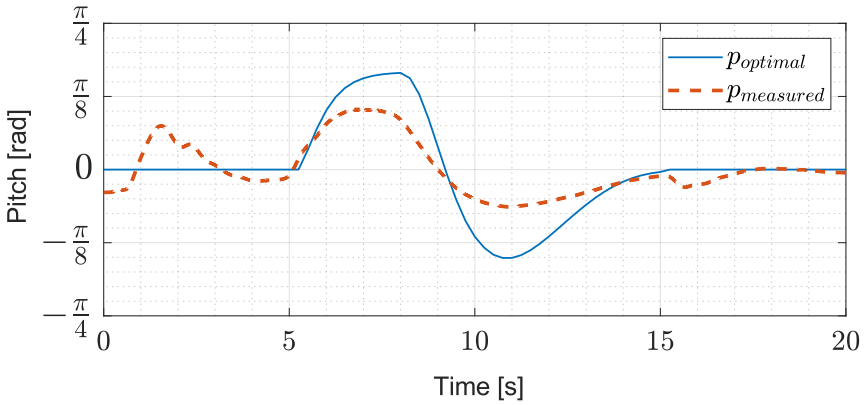
The measured elevation follows the general shape of the optimal trajectory, but doesn't reach the required height. There can be many reasons for this, but generally they are the fault of an inaccurate model. For example, pitch will result in less lift, which the model is unaware of due to decoupling. Another is that being closer to the table gives more thrust because the table provides air resistance, and flying higher thus requires more power.

3.4 Decoupled model

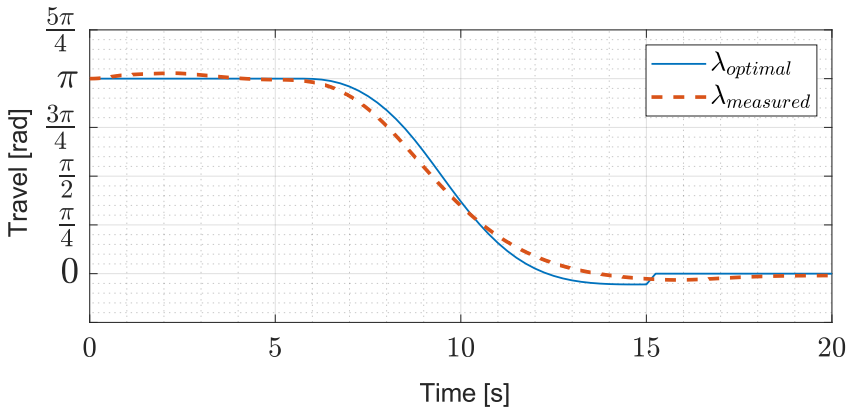
The linearization of the model has removed any connection between elevation and pitch/travel. This is of course not true, as the portion of the rotors' thrust useful as lift is proportional to the pitch angle. The effects of the inaccurate model, is that the system fails to follow the optimal calculated trajectory, most notably in regards to elevation.

As can be clearly seen in the equations of motion (eq. 1), the elevation dynamics do not consider how the downward thrust vector is decreased by the rotors' pitch. Thus, as the helicopter is in transit to its travel reference point, the elevation controller is getting less returns than expected on its inputs – resulting in the disparancy of figure 10 c) and the failure to clear the "obstacle" that is the nonlinear constraint, eq. 8.

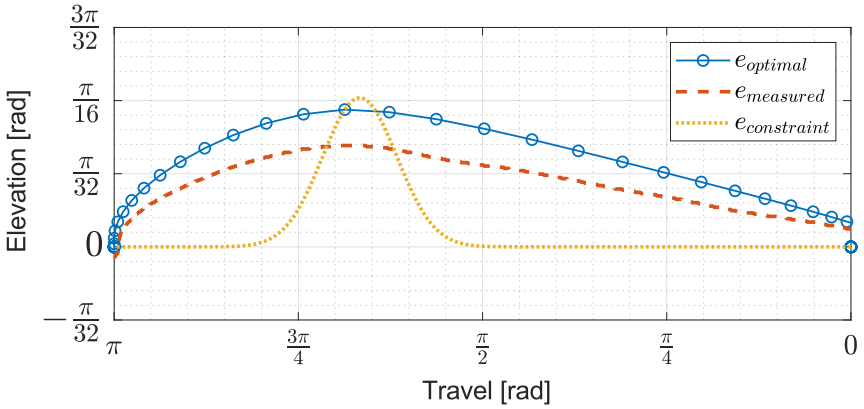
One solution would be to simply not linearize, and instead use the full nonlinear model in the open-loop optimization problem. However, LQR



(a) Optimal vs measured pitch.



(b) Optimal vs measured travel.



(c) Optimal vs elevation, as well as the nonlinear elevation constraint.

Figure 10: The measured values for pitch, travel and elevation compared to the optimal open-loop trajectories.

does not accept a nonlinear model, so it is uncertain how effective this would be.

Another conceivable improvement is using some kind of "quick fix" in Simulink, such as adding a certain amount of power once the pitch exceeds a certain level, to counteract the drop in lift. Alternatively, one could attempt to re-couple the pitch and elevation states by implementing a model for how the pitch affects the effectiveness of inputs in the elevation dynamics.

Perhaps the simplest solution would be to set even stricter constraints on pitch. This would make the system slower in terms of travel, but circumvent the issue entirely by keeping the system model closer to its linearization point.

However, we found the overall best solution to be to artifically increase the height of the obstacle. Essentially we add a margin of error to the constraint, to make sure that the real helicopter will clear the "real" original constraint. This option is discussed more in the last part of section 3.6.

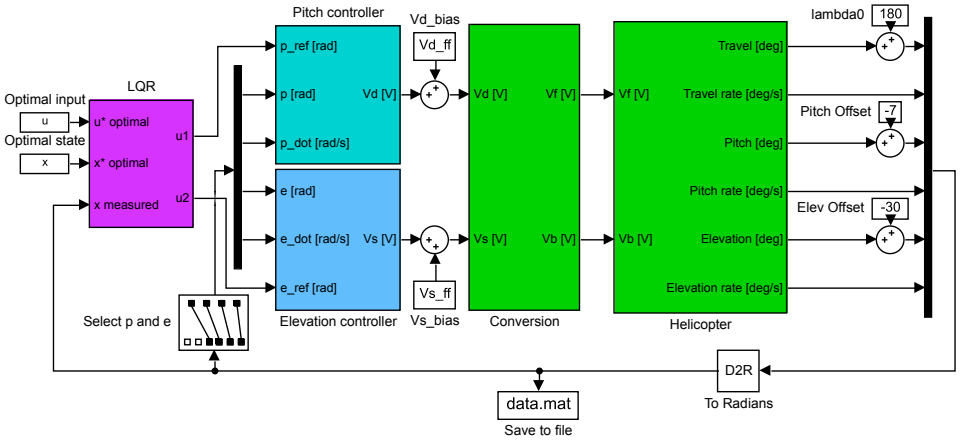
3.5 MATLAB and Simulink

Figure 11 shows the simulink diagram used. It is essentially the same as the previous task, except that the LQR now acts on all six states.

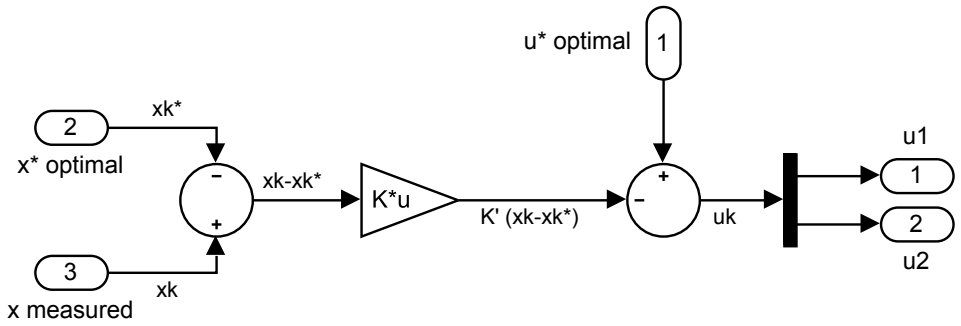
Listing 3 shows the Matlab code used. The difference from the previous task is the addition of nonlinear constraints, which requires the use of an SQP solver, *fmincon()*. The nonlinear constraints are implemented as a function at the bottom, nonLinCon().

Listing 3: Task 4 Matlab code.

```
% TTK4135 - Helicopter lab
% Template spring 2018, Andreas L. Flaten

%% Initialization

init01      % Load helicopter parameters

global nx N % To be accessed by constraint function

% Continuous model
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];
```



(a) Full simulink diagram.



(b) Simulink LQR subsystem.

Figure 11: The simulink system. The main difference from the previous task is that u is now a vector of two inputs, p_{ref} and e_{ref}.

```
17 Bc = [0  0; 0  0; 0  0; K_1*K_pp  0; 0  0; 0  K_3*K_ep];
18
19 % Number of states and inputs
20 nx = width(Ac);          % Number of states
21 nu = width(Bc);          % Number of inputs
22
23 % Time horizon and initialization
24 N  = 40;                 % Time horizon for states
25 % N = 60 % For optional exercise
26 Nx = N*nx;               % Total state variables
27 Nu = N*nu;               % Total input variables
28 Nz = N*nx+N*nu;          % Total number of optim. vars
29 z0 = ones(Nz,1);         % Initial guess
30
31 % Discrete model by forward euler
```

```
32 Dt = 0.25;
33 Ad = Dt*Ac + eye(width(Ac));
34 Bd = Dt*Bc;
35
36 % Initial value
37 lambda0 = pi;
38 x0 = [lambda0; 0; 0; 0; 0; 0];
39
40 %% Generate bounds
41 ul = [-30*pi/180; -inf]; % Pitch lower bound
42 uu = [30*pi/180; inf];   % Pitch upper bound
43
44 xl = -Inf*ones(nx,1);    % Lower bound (assume none)
45 xu = Inf*ones(nx,1);     % Upper bound (assume none)
46 xl(3)   = ul(1);         % Lower bound on pitch
47 xu(3)   = uu(1);         % Upper bound on pitch
48
49 % For optional exercise: Constraints on max
50 % travel and elevation rate.
51 % maxElevRate = 0.05;
52 % maxTravRate = 0.4;
53 % xl(2) = -maxTravRate;
54 % xu(2) = maxTravRate;
55 % xl(6) = -maxElevRate;
56 % xu(6) = maxElevRate;
57
58 [zlb,zub] = gen_constraints(N,N,xl,xu,ul,uu);
59 zlb(Nz)  = 0;   % Last input zero
60 zub(Nz)  = 0;   % Last input zero
61
62 %% Objecitve function weights
63 Q = zeros(nx);
64 Q(1,1) = 1;          % Travel
65 q = diag([1 1]);     % Weight on input
66 G = gen_q(Q,q,N,N); % Generate G matrix
67 c = zeros(Nz,1);     % No linear term
68
69 phi = @(z) z'*G*z;  % Cost function
70
71 %% Equality constraints - System model
72 Aeq = gen_aeq(Ad,Bd,N,nx,nu);
73 beq = zeros(size(Aeq,1),1);
74 beq(1:nx) = Ad*x0;
75
```

```
76 %% Solve problem using fmincon
77 opt = optimoptions('fmincon','Algorithm','sqp','
       MaxFunEvals',40000);
78 z = fmincon(phi,z0,[],[],Aeq,beq,zlb,zub,@nonLinCon,
       opt);
79
80 %% Extract control inputs and states
81 u1 = [z(Nx+1:nu:Nz);z(Nz-1)];
82 u2 = [z(Nx+2:nu:Nz);z(Nz)];
83 x1 = [x0(1);z(1:nx:Nx)];
84 x2 = [x0(2);z(2:nx:Nx)];
85 x3 = [x0(3);z(3:nx:Nx)];
86 x4 = [x0(4);z(4:nx:Nx)];
87 x5 = [x0(5);z(5:nx:Nx)];
88 x6 = [x0(6);z(6:nx:Nx)];
89
90 num_variables = 5/Dt;
91 zero_padding = zeros(num_variables,1);
92 unit_padding  = ones(num_variables,1);
93
94 % Add five second padding
95 u1  = [zero_padding; u1; zero_padding];
96 u2  = [zero_padding; u2; zero_padding];
97 x1  = [lambda0*unit_padding; x1; zero_padding];
98 x2  = [zero_padding; x2; zero_padding];
99 x3  = [zero_padding; x3; zero_padding];
100 x4  = [zero_padding; x4; zero_padding];
101 x5  = [zero_padding; x5; zero_padding];
102 x6  = [zero_padding; x6; zero_padding];
103
104 % Create timeseries
105 t = 0:Dt:Dt*(length(u1)-1);
106 X = [t' x1 x2 x3 x4 x5 x6];
107 U = [t' u1 u2];
108
109 %% LQR setup
110 Rlqr = 1;
111 Qlqr = diag([10 2 0.2 0.1 10 5]);
112 Klqr = dlqr(Ad,Bd,Qlqr,Rlqr);
113
114 %% Plotting
115 figure(3)
116 subplot(4,2,1),stairs(t,u1),grid,ylabel('u - pitch')
117 subplot(4,2,2),stairs(t,u2),grid,ylabel('u - elev')
```

```
118 subplot(4,2,3),plot(t,x1','-mo'),grid,ylabel('travel')
119 subplot(4,2,4),plot(t,x2,'-mo'),grid,ylabel('r')
120 subplot(4,2,5),plot(t,x3','-mo'),grid,ylabel('pitch')
121 subplot(4,2,6),plot(t,x4','-mo'),grid,ylabel('pdot')
122 subplot(4,2,7),plot(t,x5','-mo'),grid,xlabel('tid (s)'
    ),ylabel('elevation')
123 subplot(4,2,8),plot(t,x6','-mo'),grid,xlabel('tid (s)'
    ),ylabel('elev rate')
124
125 %% Nonlinear Constraint function
126 function [c,ceq] = nonLinCon(z)
127     alpha = 0.2;
128     % alpha = 0.3; % Optional exercise
129     beta = 20;
130     lambdaT = 2*pi/3;
131     global nx N
132     c = zeros(N,1);
133     for k=1:N
134         lambda_k = z((k-1)*nx + 1); % Each trav. var.
135         elev_k = z((k-1)*nx + 5);   % Each elev. var.
136         c(k) = alpha*exp(-beta*(lambda_k-lambdaT)^2) -
                elev_k;
137     end
138     ceq = [];
139 end
```

3.6 Optional exercise

We experimented with adding constraints on the travel rate $\dot{\lambda}$ and elevation rate $\dot{e}$, as these are states we are interested in controlling towards reference values. Recalling the system's equations of motion, the travel acceleration is modelled as being proportional to the pitch. Therefore, constraining it is equivalent to putting potentially stricter constraints on pitch, the usefulness of which was discussed in section 3.4. Brief experimentation confirmed that constraining $\dot{\lambda}$ yielded the predicted benefits of further constrainting pitch.

As for the elevation rate, the effects of its constraints can be illustrated by beholding the plotted optimal trajectories in Figure 12; in the unconstrained case, the trajectory mimics the nonlinear elevation constraint. In the constrained case, $\dot{e}$ becomes saturated, and the consequent elevation trajectory takes on a straighter shape.

The reason for this is simple: Elevation rate equals the helicopter's maximum

climbing speed – and as this climbing speed is restricted, the margins for clearing the "obstacle" in time shrinks. The optimizer is forced to abandon its preferred trajectory, in order to clear the obstacle, resulting in the unusual shape.

If the contraints on elevation rate are too strict, the helicopter will simply not be able to ascend quickly enough to avoid the obstacle – this will manifest as the solver, *fmincon*, not converging on a solution. One concludes that there is no discernible benefit to applying constraints to the elevation rate in this case, especially since in our model, elevation is decoupled from the other state groups.

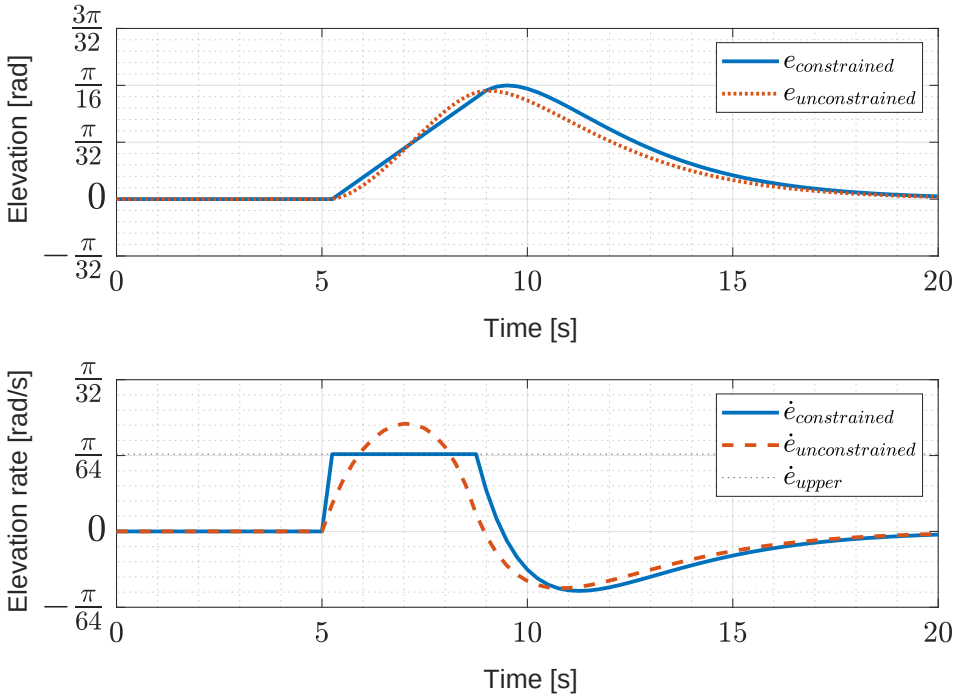


Figure 12: Comparison between optimal elevation trajectories, with and without constraints. Here, the upper bound of $\dot{e}$ is set to 0.05.

In a second experiment, we tried increasing the nonlinear constraint. The idea is that, if the constraint represents a physical obstacle, we need to clear it. However, the experiments show that the helicopter is not able to follow the required trajectory. By inspection of figure 10c, the trajectory seems to be about 1/3 too low. So far we have discussed the causes and possible solutions. However, we can also simply accept that there will be a certain amount of error, but compensate by increasing the constraints by a similar amount.

We therefore increased the nonlinear constraint by changing $\alpha = 0.2 \rightarrow 0.3$.

The results are shown in figure 13. Here, we see that the helicopter actually manages to clear the original constraint/obstacle, by using a larger constraint in the calculations. In conclusion, this seems to be the most effective way of ensuring that the original constraint/obstacle is cleared.

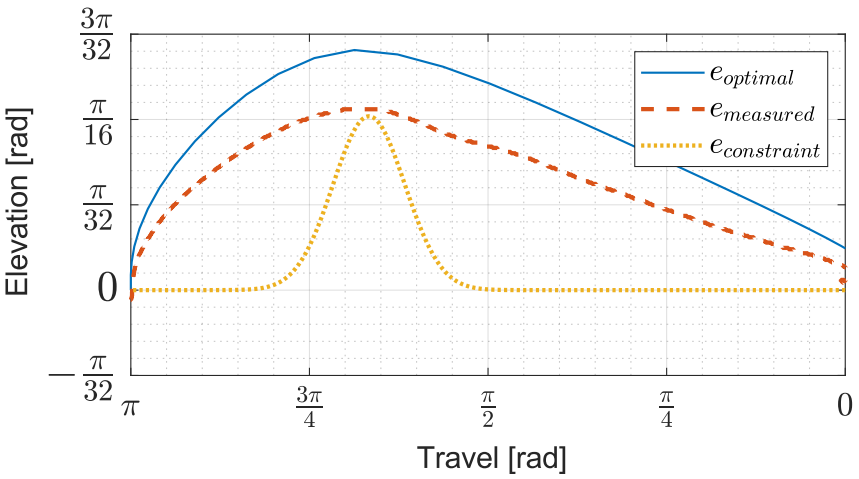


Figure 13: Optimal and measured elevation when using $\alpha = 0.3$ in the constraint function, compared to the original constraint where $\alpha = 0.2$.

References

[1] Department of Engineering Cybernetics. *TTK4115 Linear System Theory, Helicopter Lab Assignment Text*. NTNU, 2021.

[2] Department of Engineering Cybernetics. *TTK4135 Optimization and Control, Helicopter Lab Assignment Text*. NTNU, 2021.