TTK4135 - Optimization and Control
Helicopter Lab Report

Group 1

Tran, Mariell Hoff - 544070
Hoff, Ola Kristoffer - 526024

20^{th} April, 2023

Contents

1 10.2 - Optimal Control of Pitch/Travel without Feedback

1.1 The continuous model

The system is described by the given model:

$$\begin{aligned} \dot{x} &= A_c x + B_c u \\ x &= \begin{bmatrix} \lambda & r & p & \dot{p} \end{bmatrix}^\top \\ u &= p_c \end{aligned}$$

Here we derive the model on continuous time state space form:

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

$$\dot{x} = \begin{bmatrix} \dot{\lambda} \\ \dot{r} \\ \dot{p} \\ \ddot{p} \end{bmatrix}, \quad u = p_c$$

$$A_c = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} \end{bmatrix}$$

$$B_c = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1 K_{pp} \end{bmatrix}$$

$$\dot{x} = \begin{bmatrix} \dot{\lambda} \\ \dot{r} \\ \dot{p} \\ \ddot{p} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -k_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1 K_{pp} \end{bmatrix} p_c = A_c x + B_c u$$

Here we have a model of the helicopter, but only it's: travel, travel rate, pitch and pitch rate. As stated in the task description the elevation is ignored and assumed to be constant at this stage.

In figure 1 we see the control hierarchy. Here our mathematical models are used in the "Model-based optimazation". Here we solve the optimal path based on the start and end states. Then the input sequence to obtain this is sent to the pitch and elevation controllers (here elevation controller just keeps elevation constant). Then the controllers calculate the values we need to get the given value in the input sequence. The error, which will accumulates while running, will not be taken into account. Then helicopter will "think" it is where the input sequence needs it to be.

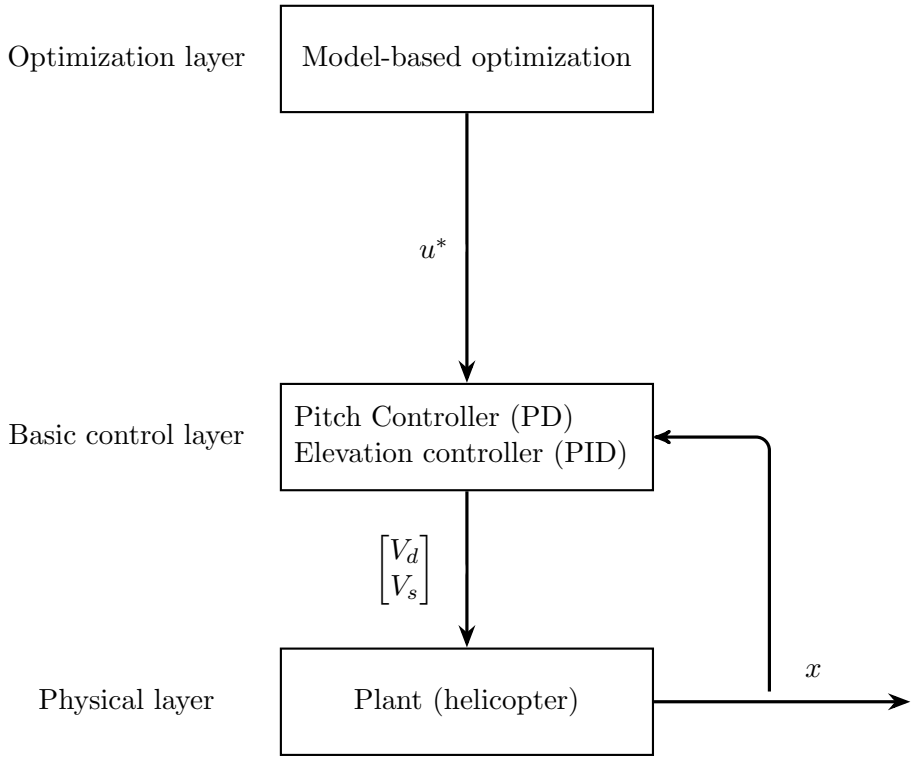


Figure 1: Illustration of the layers in the control hierarchy used in Section 10.2. Copied from the *LabExercises 2023.pdf* (figure 7).

1.2 The discretized model

Using the *Forward Euler* method we derive the discrete time state space form of the model:

$$\begin{aligned}
\dot{x} &= A_c x + B_c u \\
\dot{x}_k &= \frac{x_{k+1} - x_k}{\Delta t}, \quad \text{Forward Euler method} \\
\dot{x}_k &= A_c x_k + B_c u_k \\
\frac{x_{k+1} - x_k}{\Delta t} &= A_c x_k + B_c u_k \\
x_{k+1} &= x_k + \Delta t A_c x_k + \Delta t B_c u_k \\
x_{k+1} &= \underbrace{(\mathbb{I} + \Delta t A_c)}_{A_d} x_k + \underbrace{\Delta t B_c}_{B_d} u_k \\
x_{k+1} &= A_d x_k + B_d u_k
\end{aligned}$$

1.3 The open loop optimization problem

We are trying to optimise the trajectory of the helicopter. This is done by minimising the function:

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

We were to use the *quadprog*-function in *MATLAB*, it requires the problem to be a QP on standard form.

The standard form is stated as:

$$\underset{z \in \mathbb{R}^n}{min} \ \frac{1}{2} z G z^\top, \quad \text{s.t. } A_{eq} z = b_{eq} \tag{2}$$

We can start by collecting all the variables in one variable, z:

$$z = (x_1^\top, \ldots, x_N^\top, u_0^\top, \ldots, u_{N-1}^\top)$$

Recognise that the objective function is currently on the form:

$$f(z) = \sum_{i=0}^{N-1} \frac{1}{2} x_{i+1}^\top Q x_{i+1} + \frac{1}{2} u_i^\top R u_i$$

Since we have four states; travel, travel rate, pitch and pitch rate, our x input is a vector with four elements with the λ being the travel. Here all other variables are zero.

$$x_i = \begin{bmatrix} \lambda_i - \lambda_f \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

This gives us the Q as:

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

From the function (1) we see that $R = 2q$. We can now write the function on the form 2, with G:

$$G = \begin{bmatrix} Q_1 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & Q_N & \ddots & & \vdots \\ \vdots & & \ddots & R_0 & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & R_{N-1} \end{bmatrix}$$

Here $N = 100$.

The constraints is expressed as:

$$A_{eq} = \left[\begin{array}{ccccc|ccccc} I & 0 & \cdots & \cdots & 0 & -B_0 & 0 & \cdots & \cdots & 0 \\ -A_1 & I & \ddots & & & 0 & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 & \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -A_N & I & 0 & \cdots & \cdots & 0 & -B_N \end{array}\right]$$

and

$$b_{eq} = \begin{bmatrix} A_0 x_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Here $A_i = A_d$ and $B_i = B_d$ from 1.2. The constraint on the pitch is included in the constraints above.

To answer why we need the extra constraint on the pitch let's consider the case of not having it. Since the objective is to change the travel (from π to 0), and the direction and amount of force is directly a function of the pitch, we don't want the helicopter to flip sideways full throttle and half way do a 180 degree turn and slow down. This would minimise our objective function the most. However, we want a smoother and more secure pitch, hence we limit it to a few degrees.

The pitch reference angle, p_c, does not need to be limited, as long as the general pitch limitation takes into account the reference pitch.

In listing 1 we see the code for the implementations of the constraints. In listing 2 we see the code for the actual solving of the problem.

Listing 1: *MATLAB* code for setting the constraints.

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

xl = -Inf*ones(mx,1);   % Lower bound on states
xu = Inf*ones(mx,1);    % Upper bound on states
xl(3) = ul;             % Lower bound on state x3
xu(3) = uu;             % Upper bound on state x3

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

Listing 2: *MATLAB* code for solving the QR.

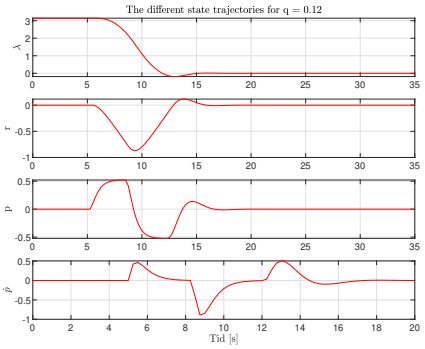
```
%% Generate system matrixes for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu);   % Generate A, hint: gen_aeq
beq = zeros(1, 400);            % Generate b
beq(1,1) = pi;
%% Solve QP problem with linear model
[z,lambda] = quadprog(Q, [], [], [], Aeq, beq, vlb, vub);
```

1.4 The weights of the optimization problem

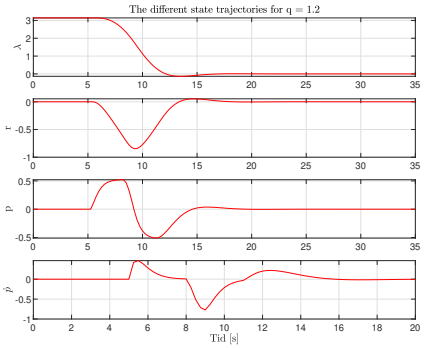
The calculated optimal state trajectories from the optimisation problem with the different q values can be seen in figures; 2(a), 2(b) and 2(c).

As seen on the plots the weighting effects the speed of convergence of the states. This is because q decides the input use of the controller a smaller q penalises the use of input less while a larger q penalises the use of input more.

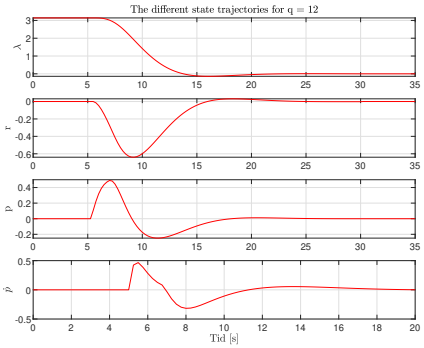
The differences are shown more clearly in figure 3. In the case of this system $\mathbf{u} = \mathbf{p_{ref}}$ so the input trajectories is the same as the pitch reference. The plot shows clearly that a smaller q makes the system use more input and is more volatile, while a larger q has a more gentle slowly convergence due to it's more restricted use of input.



((a)) Theoretical state trajectories with q = 0.12.



((b)) Theoretical state trajectories with q = 1.2.



((c)) Theoretical state trajectories with q = 12.

Figure 2: Optimal trajectories for all states given different values for q.

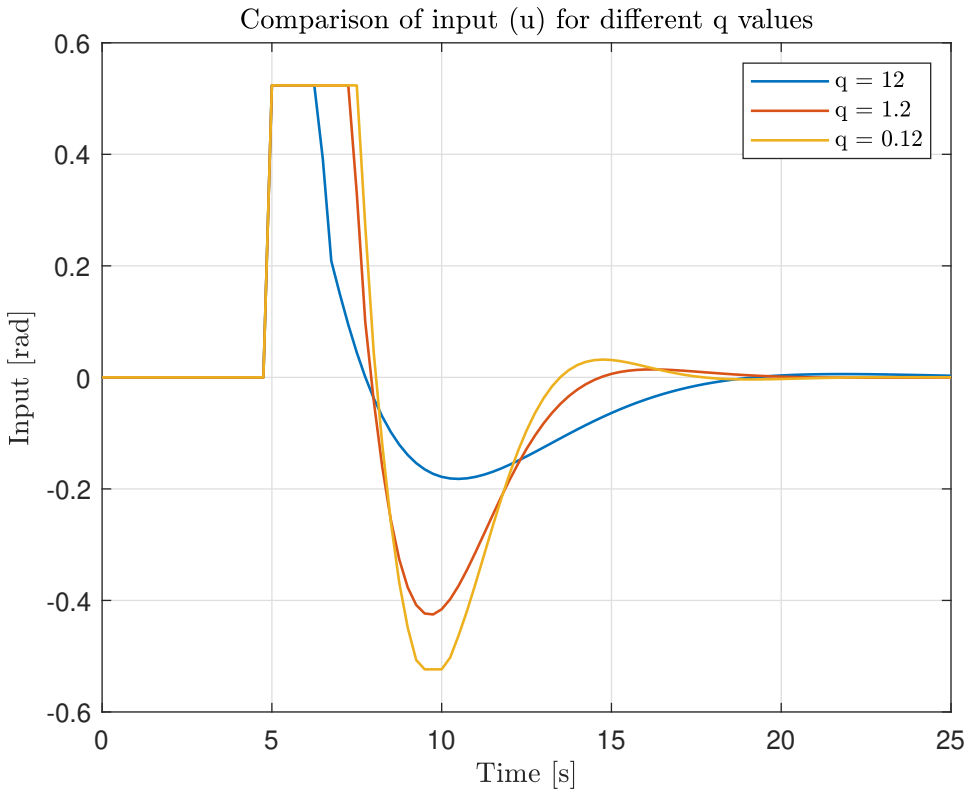


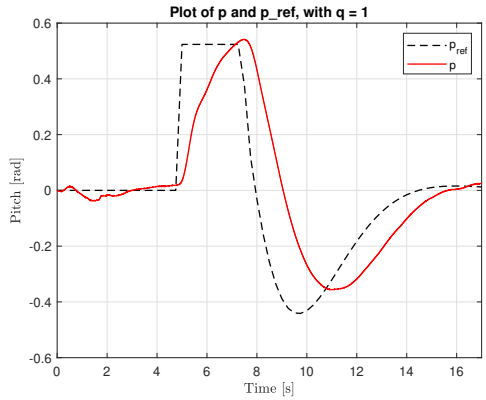
Figure 3: Comparison of the input's (u/p_{ref}) trajectories for different q values.

1.5 Experimental results

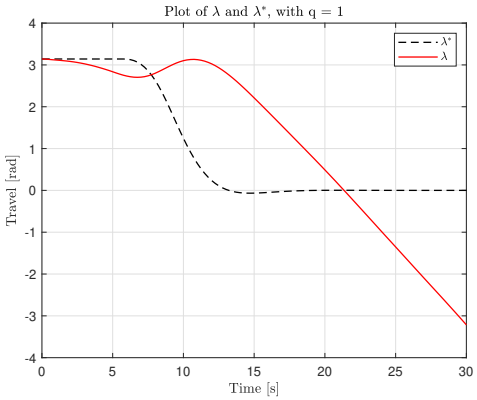
Disclaimer: we used the helicopter station 2 for this task and the results seem reasonable. However, we had to do the rest of the lab sessions on different stations due to the fact that it did not work on station 2. We are unsure what effects station 2 might have had on the results of this first task.

A plot of the desired optimal $\mathbf{p_{ref}}$ and physical helicopter's trajectory is shown in figure 4(a). The physical system is seen to be slower than the calculated trajectory. This can be attributed to slowness in the physical system that is not accounted for in the modelling, such as air resistance and slowness in changing direction. The physical system is also close to $\mathbf{x_f}$ but doesn't converge entirely. This deviation can come from the open loop control. The system has no feedback from actual measurements during the flight so there is no way to adjust accordingly. Therefore it is difficult to achieve the desired result.

The travel, seen in figure 4(b), is far from good. It does not respond well in the start, it is a bit slow to begin as the pitch. However, when it starts moving it never stops, it is in a constant tilt which makes the helicopter go around in circles, not to fast, but as we can see it does not converge. This too is most likely due to the real-world variables not included in the model, and the lack of adjusting to the current state. If feedback had been present we could have stabilised the helicopter at the wanted travel within a interval of certainty.



((a)) Plot comparing the optimal input p_{ref} with the actual pitch trajectory p.



((b)) Plot comparing the optimal travel λ^* and the actual travel trajectory λ.

Figure 4: Optimal and observed trajectories for pitch and travel with optimised input sequence.

1.6 MATLAB and Simulink

1.6.1 MATLAB code

Listing 3: *MATLAB* code for task 10.2.

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

% Discrete time system model. x = [lambda r p p_dot]'
delta_t = 0.25; % sampling time
A1 = [1 delta_t 0 0;
      0 1 -delta_t * K_2 0;
      0 0 1 delta_t;
      0 0 -K_1*K_pp*delta_t 1-K_1*K_pd*delta_t];
B1 = [0; 0; 0; delta_t*K_1*K_pp];

% Number of states and inputs
mx = size(A1,2); % Number of states (number of columns in A)
```

```
mu = size(B1,2); % Number of inputs(number of columns in B)

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

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

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

xl = -Inf*ones(mx,1);         % Lower bound on states (no bound)
xu = Inf*ones(mx,1);          % Upper bound on states (no bound)
xl(3) = ul;                   % Lower bound on state x3
xu(3) = uu;                   % Upper bound on state x3

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

% Generate the matrix Q and the vector c (objecitve function
   weights in the QP problem)
Q1 = zeros(mx,mx);
Q1(1,1) = 1;                              % Weight on state x1
Q1(2,2) = 0;                              % Weight on state x2
Q1(3,3) = 0;                              % Weight on state x3
Q1(4,4) = 0;                              % Weight on state x4
P1 = 12;                                  % Weight on input
Q = gen_q(Q1,P1,N,M);                     % Generate Q, hint: gen_q
c = 0; % Generate c, this is the linear constant term in the QP

%% Generate system matrixes for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu);   % Generate A, hint: gen_aeq
beq = zeros(1, 400);            % Generate b
beq(1,1) = pi;
%% Solve QP problem with linear model
tic
[z,lambda] = quadprog(Q, [], [], [], Aeq, beq, vlb, vub); % hint:
    quadprog. Type 'doc quadprog' for more info
t1=toc;
```

1.6.2 Simulink diagram

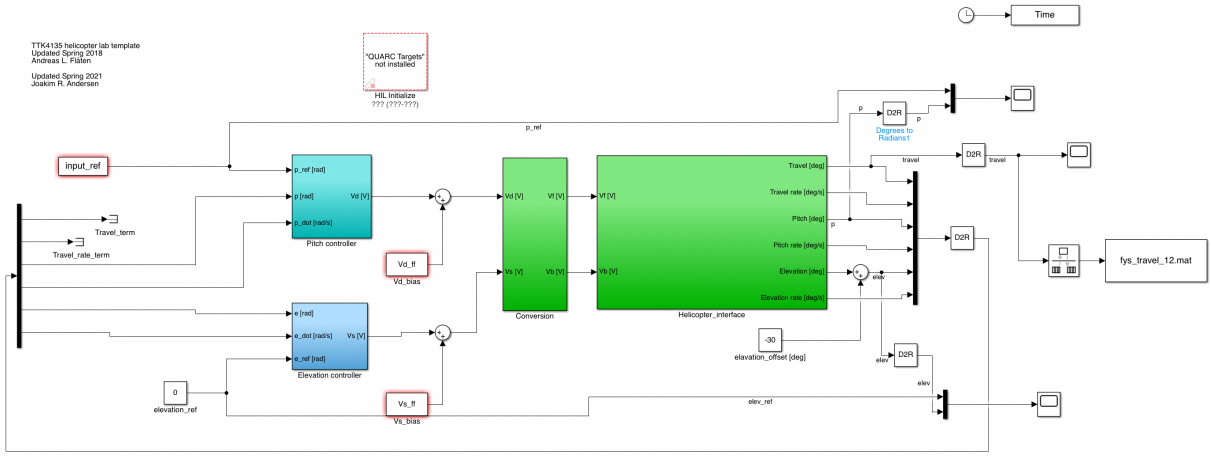


Figure 5: Simulink diagram from lab day 2.

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

2.1 LQ controller

The LQ controller stand for linear quadratic controller and is a state feedback controller. It uses the weighting of **Q** and **R** matrices to minimise a quadratic criteria. **Q** and **R** are weight matrices for states and input, they indicated how much we want to penalise the use of power to manipulate the variables. From the optimal trajectory calculated in 1.4 we have an optimal input u^* and optimal trajectory x^*. The LQ controller introduces optimal feedback by the manipulated variable u, as seen in the control hierarchy in figure 6. The manipulated variable u can be formulated as,

$$u = u^* - \mathbf{K}(x - x^*). \tag{3}$$

The choice of the feedback gain **K** is determined by the weight matrices that minimises the quadratic linear function,

$$J = \sum_{i=0}^{\infty} \Delta x_{i+1}^\top Q \Delta x_{i+1} + \Delta u_i^\top R \Delta ui, \quad Q \geq 0, R \geq 0. \tag{4}$$

The tuning of the **Q** and **R** matrices was based on the want for an optimal travel trajectory from λ_0 to λ_f, therefore the used matrices were defined as,

$$\mathbf{Q} = \begin{bmatrix} 50 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$
$$\mathbf{R} = 1.$$

The general idea was to penalise derivations in travel and to give leeway to pitch so that it could easily adjust. The resulting feedback gain **K** could be computed by solving the *Ricatti equation* resulting in a gain equal to

$$\mathbf{K} = \begin{bmatrix} -3.57 & -8.23 & 3.76 & 0.96 \end{bmatrix}.$$

The feedback gain will give different gains to the deviations, $\Delta x = x^* - x$ depending on weighting in the **Q** matrix and the system model, to set the system input u.

Practically in *MATLAB* this is done in a few lines of code as shown in listing 4.

Listing 4: *MATLAB* code solving for K.

```
A1 = [1 delta_t 0 0;
      0 1 -delta_t * K_2 0;
      0 0 1 delta_t;
      0 0 -K_1*K_pp*delta_t 1-K_1*K_pd*delta_t];
B1 = [0; 0; 0; delta_t*K_1*K_pp];

v = [50,1,10,1];
Q_2 = diag(v);
R = 1;

[K,P,e] = dlqr(A1,B1, Q_2,R);
```

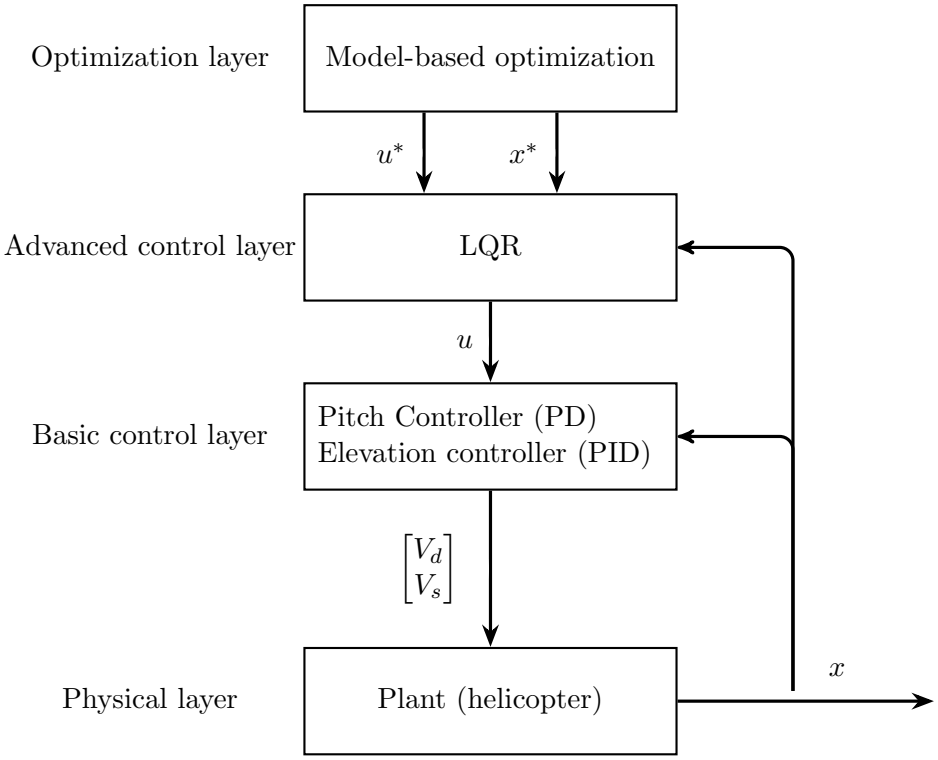


Figure 6: Illustration of the layers in the control hierarchy used in Sections 10.3 and 10.4. Copied from the *LabExercises 2023.pdf* (figure 8).

2.2 Model Predictive Control

Model predictive control is a closed loop controller that at each time-sampling of the process, solves the open loop optimisation problem. Instead of only solving it once as derived in 1.3. It uses the current input and output measurements in addition to the states to calculate the optimal trajectory $\mathbf{x}^*$. Then it will only send the first optimal calculated input $\mathbf{u(t)} = \mathbf{u_0^*}$ in the sequence to the process before it repeats the process and calculates the next optimal input from the sampling at the next time-step. The structure figure 6 with the implemented MPC controller is shown in figure 7.

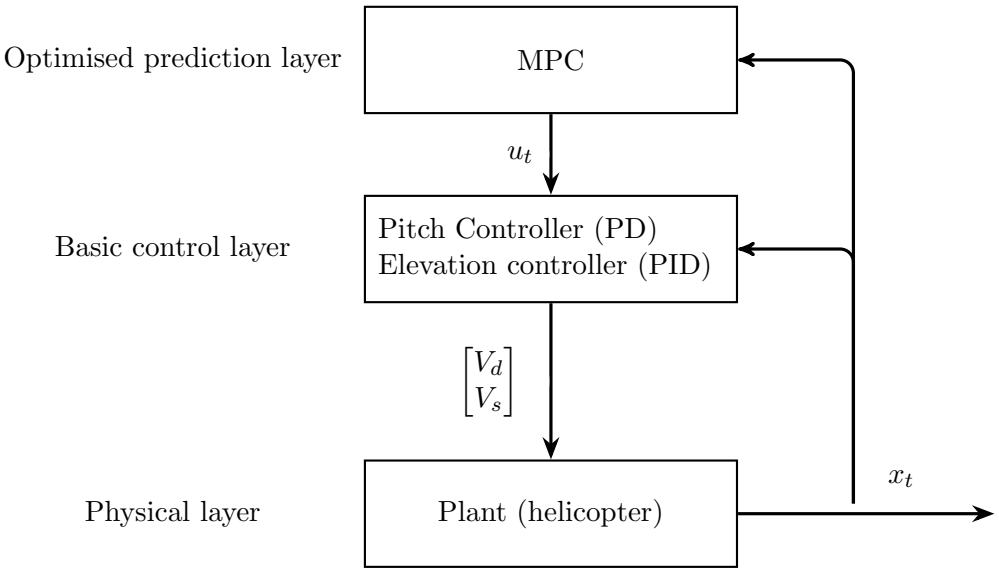


Figure 7: Illustration of the layers in the control hierarchy if an MPC were to replace the prediction and LQR method.

Compared to the LQ controller an MPC controller would be more robust against disturbances since it updates the prediction and optimal path for every time-step. One disadvantage of this is that the MPC requires more computational resources, to calculate $\mathbf{x}^*$ and $\mathbf{u}^*$ repeatedly. This can be improved by implementing input blocking that can reduce run times.

2.3 Experimental results

Printouts of data from relevant experiments (plots). Discussion and analysis of the results. Answer 10.3.2.5 here.

The used Q and R in the resulting plot was:

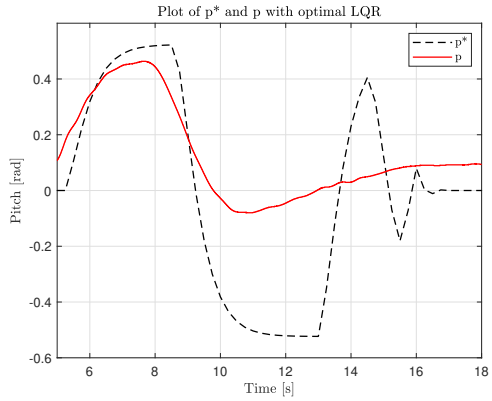
$$Q = \begin{bmatrix} 50 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

$$R = 1$$

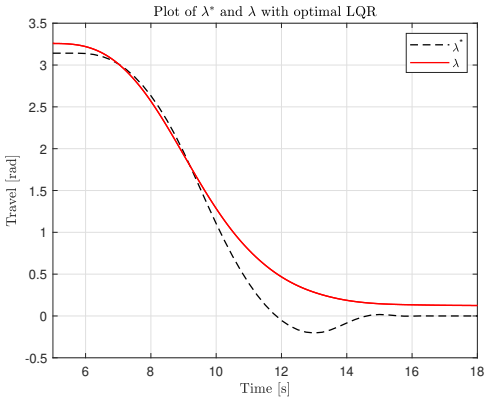
Looking at figure 8(a) first; we can see that the observed pitch is following the optimal trajectory decently until the optimal wants a dip in the pitch. Here our result flattens out and actually increase a bit. The offset our resulting pitch has in the end is due to real world implications of it having to tilt a bit to keep travel constant, make the helicopter stand in one place.

Our general idea for the weights of the Q and R was to focus more on the travel being correct, with respect to the predicted optimal path, such that the pitch would easier adjust as needed to keep the helicopter in route to its destination. This was fairly successful as we can see in figure 8(b).

Here the resulting travel is very close to the optimal travel until the optimal overshoots a bit. Our result does not overshoot, but rather slows down significantly more and stabilises itself. The offset our result has from the wanted destination value of $\lambda_f = 0$ is small and probably a result of the real-world variance.



((a)) Plot of pitch and optimal pitch with feedback in the system.



((b)) Plot of travel and optimal travel with feedback in the system.

Figure 8: Optimal and observed trajectories for pitch and travel with optimised input sequence.

Comparing these results to the ones in section 1.5 we see some obvious changes in pitch. This is mostly due to the fact that we have weighted travel as much more important to keep correct, meaning that pitch will adjust to correct for any deviation from the optimal travel trajectory. Travels trajectory is much closer to the wanted behaviour, this is a result of the systems ability to adjust from updated measurements and the ability to tune the controller to our preferred behaviour. We now see that pitch and travel both converge on a value (if not perfect), they do not diverge of to infinity. This is a great improvement over the previous results and it shows the clear benefits of a feedback system to handle accumulating errors over time.

2.4 MATLAB and Simulink

2.4.1 MATLAB code

The only changes in the *MATLAB*-code is the four code lines after the comment "LQ". All the other code is the same as in the previous section.

Listing 5: *MATLAB* code for task 10.3.

```
%% Generate system matrixes for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu);   % Generate A, hint: gen_aeq
beq = zeros(1, 400);            % Generate b
beq(1,1) = pi;

%% LQ
v = [50,1,10,1];
Q_2 = diag(v);
R = 1;

[K,P,e] = dlqr(A1,B1, Q_2,R);

%% Solve QP problem with linear model
tic
[z,lambda] = quadprog(Q, [], [], [], Aeq, beq, vlb, vub); % hint:
    quadprog. Type 'doc quadprog' for more info
t1=toc;
```

2.4.2 Simulink diagram

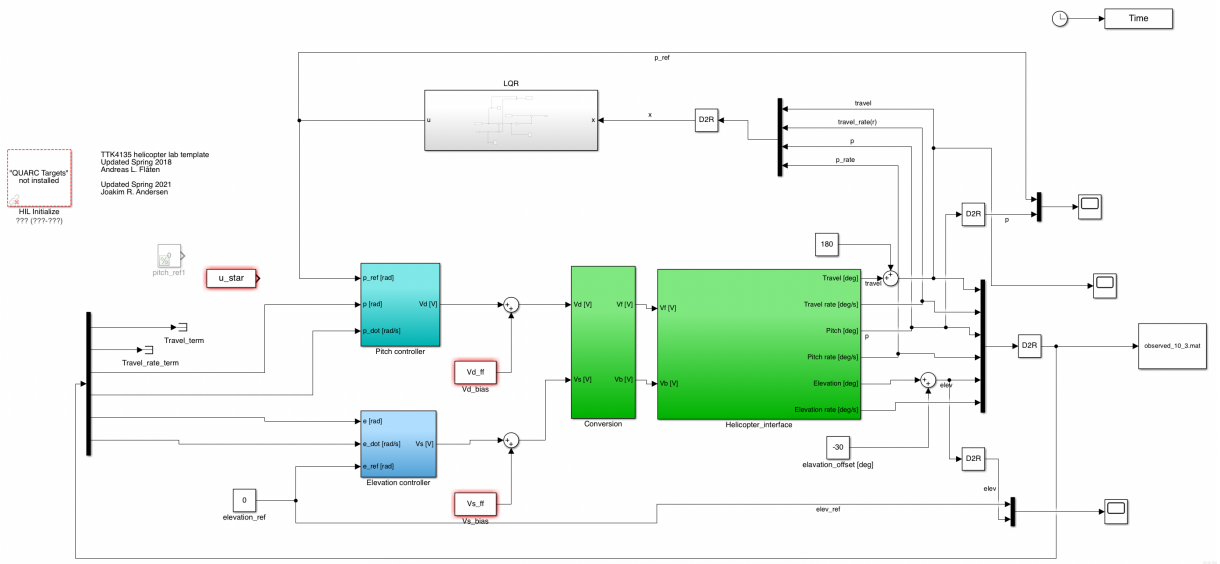


Figure 9: Simulink diagram from lab day 3.

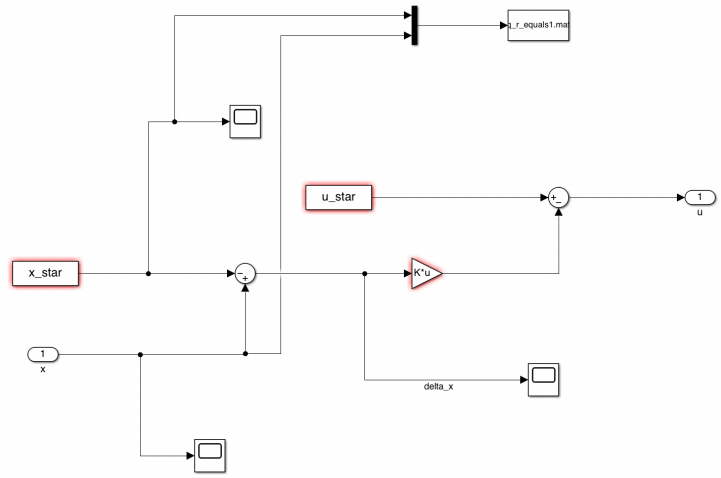


Figure 10: The inside of the LQR block in figure 9.

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

3.1 The continuous model

To include the dynamics of the elevation, the continuous state space model must include $\mathbf{e}$ and $\dot{\mathbf{e}}$. We derive it the same way as in 1.1, and get the following:

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

3.2 The discretized model

Answer 10.4.1.2 Discretized the model using forward Euler method described in 1.2 to get the discrete space time form. The discrete space time model is then given as:

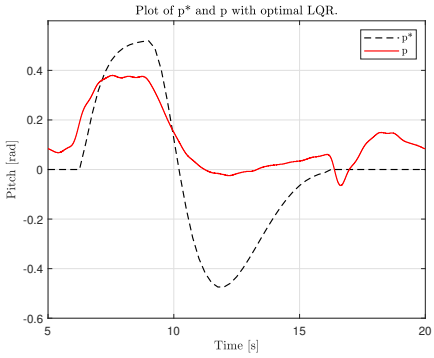
$$\mathbf{A_d} = \begin{bmatrix} 1 & \Delta t & 0 & 0 & 0 & 0 \\ 0 & 1 & -\Delta t K_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & \Delta t & 0 & 0 \\ 0 & 0 & -\Delta t K_1 K_{pp} & 1 - \Delta t K_1 K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & -\Delta t K_3 K_{ep} & 1 - \Delta t K_3 K_{ed} \end{bmatrix}$$

$$\mathbf{B_d} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \Delta t K_1 K_{pp} & 0 \\ 0 & 0 \\ 0 & \Delta t K_3 K_{ep} \end{bmatrix}$$

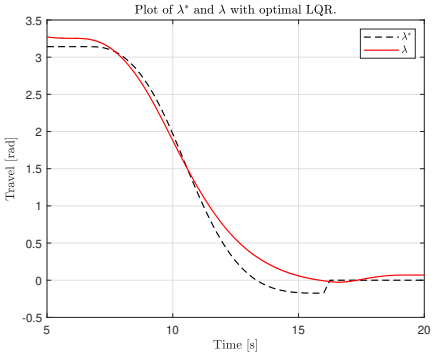
3.3 Experimental results

As we can see in figure 11 our final result is quite good. We still have the weighting such that the focus is on the physical positioning of the helicopter (travel and elevation) and giving leeway to pitch such that it can correct and adjust accordingly when travel and/or elevation deviates from the optimal trajectory.

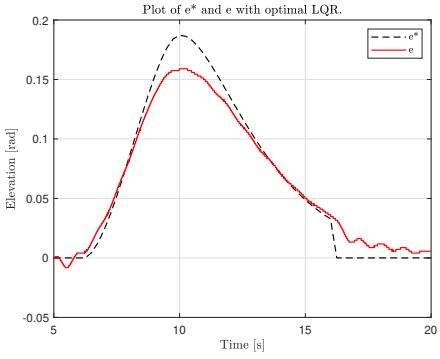
This is clear in figure 11(a) that pitch does not follow the optimal trajectory, but this was expected. In figure 11(b) and 11(c) we see very good results, the observed trajectories are very close to the optimal ones. The plots support our planed weighting, and the resulting trajectory is controlled and good.



((a)) Plot of pitch and optimal pitch with feedback in the system.



((b)) Plot of travel and optimal travel with feedback in the system.



((c)) Plot of elevation and optimal elevation with feedback in the system.

Figure 11: Observed and optimal plots of pitch, travel and elevation.

3.4 Decoupled model

In the state model elevation and pitch are completely decoupled, since we derive a linear model for elevation. In reality the relationship is sinusoidal and equal to,

$$\ddot{e} = C_p V_s \cos p + C_e \cos e.$$

The system still runs smoothly without taking into account the dynamics of which pitch and elevation effects each other. This is because the controller has access to the angular rates and can be used to control elevation despite the decoupling. One other reason is that the LQR is tuned to handle uncertainties in the modelling of the helicopter, it can therefore adjust it's inputs to take these uncertainties into account.

For improving the controller by taking into account the relationship between elevation and pitch, there are three options to discuss.

The first is to rewrite the expression of $\ddot{e}$ to the sinusoidal relationship, this will make the model be closer to reality but it will also make for a nonlinear system, which is harder to account for in the controller. You would have to implement a way to linearly approximate the non-linear behaviour for the LQ controller. This would add the need for more computational resources.

The second solution is to add the elevation and pitch relationship dynamics to the objective function. This way, when we minimise the objective function we obtain an optimal trajectory which takes into consideration the relationship between the pitch and elevation. If no real-world errors had existed this would be great. However, real-world errors do exist and the LQR will be no more fit to correct the trajectory taking the pitch-elevation-relation into consideration, meaning that over time, it would amount to the same as if we had used or original objective function.

The third is adjusting the **Q** matrix weight in elevation to take pitch changes into account and update this every few time-steps. If the elevation weight was multiplied by the cosine of the pitch this would make the weight vary in value. More precisely it would make the focus on elevation heavily weighted when the pitch is great (meaning the helicopter is "more" sideways, hence we are losing more elevation), and negligible when the pitch is small (when the helicopter is flat, we don't loose elevation). However, as in the first method, we get a non-linear system, because of the cosine expression. This is not as easily solved, and more computational heavy.

3.5 MATLAB and Simulink

3.5.1 MATLAB code

Listing 6: *MATLAB* code for task 10.4.

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

% Discrete time system model. x = [lambda r p p_dot e e_dot]'
delta_t = 0.25; % sampling time
A1 = [1 delta_t 0 0 0 0;
      0 1 -delta_t * K_2 0 0 0;
      0 0 1 delta_t 0 0;
      0 0 -K_1*K_pp*delta_t 1-K_1*K_pd*delta_t 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];

B1 = [0 0; 0 0; 0 0; delta_t*K_1*K_pp 0; 0 0; 0 delta_t*K_3*K_ep];

% Number of states and inputs
mx = size(A1,2); % Number of states (number of columns in A)
mu = size(B1,2); % Number of inputs(number of columns in B)

```

```
% Initial values
x1_0 = pi;                                  % Lambda
x2_0 = 0;                                   % r
x3_0 = 0;                                   % p
x4_0 = 0;                                   % p_dot
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
z0(1) = x1_0;

% Bounds
% Transpose these to column vector so they can be used in "
   gen_constraints"
ul = [-pi/6, -Inf]';     % Lower bound on control
uu = [pi/6, 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) = 1;                                % Weight on state x1
Q1(2,2) = 0;                                % Weight on state x2
Q1(3,3) = 0;                                % Weight on state x3
Q1(4,4) = 0;
Q1(5,5) = 0;
Q1(6,6) = 0;

% Weight on state x4
P1 = zeros(mu,mu);                          % Weight on input p_c
P1(1,1) = 1;                                % Weight on input e_c
P1(2,2) = 1;
Q = gen_q(Q1,P1,N,M);                       % Generate Q, hint: gen_q
% Generate c, this is the linear constant term in the QP

%% Generate system matrixes for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu);  % Generate A, hint: gen_aeq
beq = zeros(1, size(Aeq, 1));  % Generate b
beq(1,1) = pi;

%% LQ
```

```
v = [250,1,0.5,1,600,0.5];
Q_2 = diag(v);
R = [5,1];
R = diag(R);

K = dlqr(A1,B1, Q_2,R);

func = @(z)1/2*z'*Q*z

%% Solve QP problem with linear model
nonlcon = @elevation_con;
tic
options = optimoptions('fmincon', 'Algorithm','sqp','Display','off
    ');
[z,lambda] = fmincon(func, z0, [], [], Aeq, beq, vlb, vub, nonlcon
    , options); % hint: quadprog. Type 'doc quadprog' for more info
t1=toc;

function [c,ceq] = elevation_con(z)
    %Nonlinear constraint
    alfa = 0.2;
    beta = 20;
    lambda_t = (2*pi)/3;
    N = 40;
    c = zeros(N,1);
    for k = 1:N
        lambda = z((k-1)*6 + 1);
        e = z((k-1)*6 + 5);
        c(k) = alfa*exp(-beta*(lambda-lambda_t)^2)-e;
    end
    c;
    ceq = [];
end
```

3.5.2 Simulink diagram

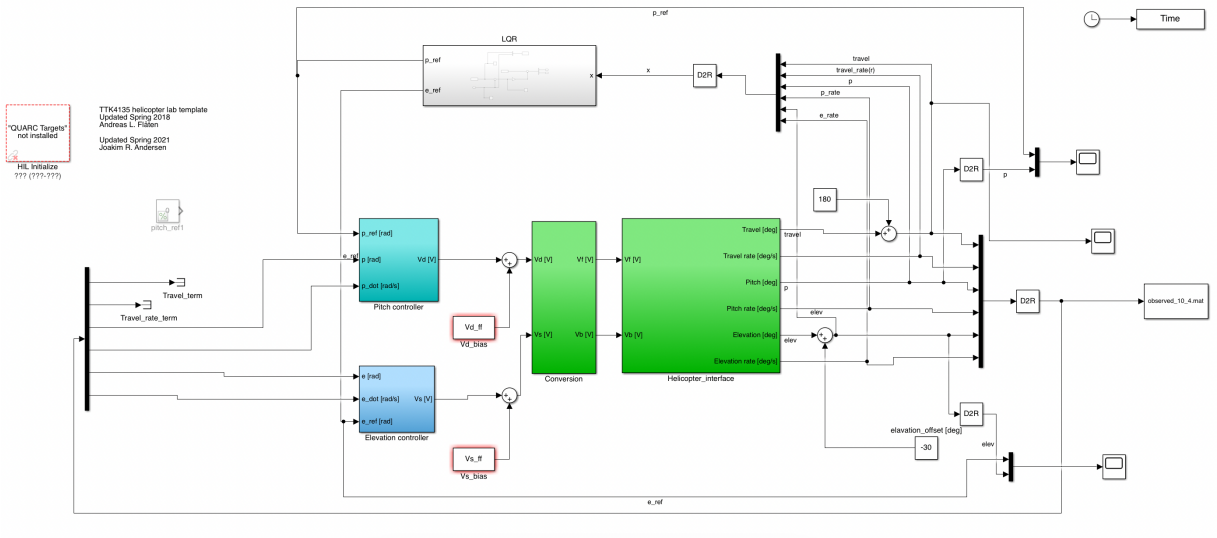


Figure 12: Simulink diagram from lab day 4.

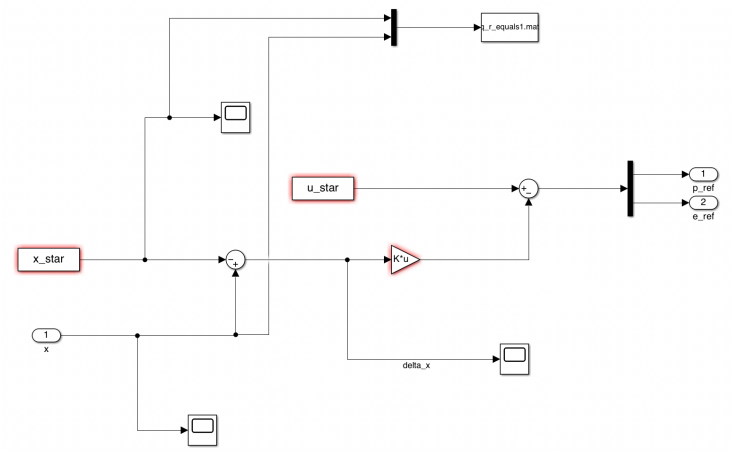


Figure 13: The inside of the LQR block in figure 12.