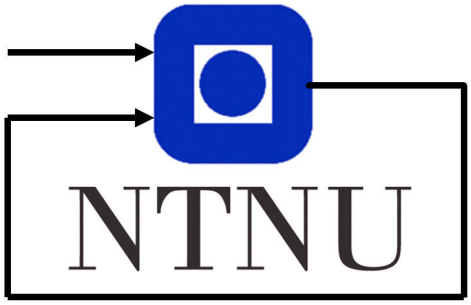
TTK4135 Helicopter labreport

Group 41
Student number: 756411
Student number: 756382

November 12, 2018



Department of Engineering Cybernetics

Contents

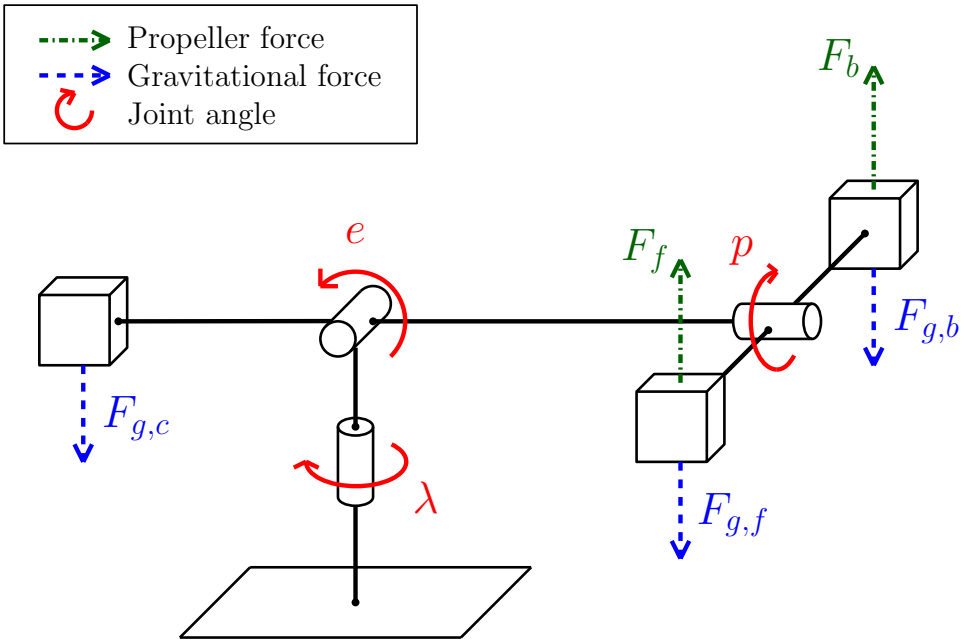


Figure 1: Force diagram representation of our helicopter.

Motivation

The goal of this assignment is to get some hands-on experience in formulating a dynamic optimization problem, discretizing and realize the problem on a computer, and finally solving it. Another goal of the assignment is to teach us some ways to implement omptimal control with and without feedback in a modern environment using Simulink and MATLAB. All of this is realized on a helicopter at one of the labs at NTNU.

This project is part of the mandatory coursework in the course TTK4135, Optimization and Control, at NTNU in the spring of 2018.

The physical setup for this project consists of a two-rotor helicopter fixed on an arm with a counterweight on the other end. This is again connected to another arm which is connected to the base. Figure fig. 1 shows a force representation of our helicopter setup. The different angles are measured with optical position sensors. The helicopter is then connected to a QuaRC-server that we use our workstation and simulink to interface with.

1 Familiarizing ourselves with the starting point of the project

We were given an initial Simulink-model of the helicopter as well as a MATLAB script with the necessary physical parameters for our masses and lengths. The first assignment was to familiarize ourselves with the handed out code. The code consisted of some helper-functions to generate constraints, and converting matrices to QP standard form, and a figure of the handed out Simulink model can be seen in fig. 13 in appendix C.

The Simulink file consists of four blocks. The first is the helicopter interface, it enables us to get measurements from our helicopter as well as apply inputs to the helicopter motors. The next block is a conversion module that converts the sum and difference of motor voltages (our controller output) into the front and back motor voltage (what the helicopter input is). Lastly we are given two mono-variable controllers, one for pitch and one for elevation.

In this project we were to treat the mono-variable pitch and elevation controllers as the basic control-layer on which to build our optimization layer. We assume the controllers to be well-tuned and well-behaved so no effort was made to further tune these controllers.

After confirming that we could compile and run the handed out files, and that the helicopter responded as expected we moved on to implement optimal control without feedback

2 Optimal control of Pitch/Travel without feedback

In this part of the exercise we will disregard elevation, that is, we assume $e = 0$ at every time-step. We will then calculate an optimal trajectory x^* and a corresponding optimal input sequence u^*. This input sequence will be implemented as setpoints for the inner controllers, but we will not feed back the measured state to correct for deviations from the optimal trajectory.

2.1 Continuous State-space form of dynamics

The closed loop inner dynamics for our helicopter with a PID elevation controller and a PD pitch controller can be seen in eq. (1), these equations were given in the assignment.

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

First, we wish to put the system on continuous state-space form as in eq. (2), with state and unit vector as given in eq. (3). The resulting system matrices are shown in eq. (4).

$$\dot{x} = A_c x + B_c u \tag{2}$$

$$x = \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \end{bmatrix}, \quad u = p_c \tag{3}$$

$$\mathbf{A_c} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} \end{bmatrix}, \quad \mathbf{B_c} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1 K_{pp} \end{bmatrix} \tag{4}$$

The model in eq. (4) represents the closed-loop dynamics of our plant, the helicopter, as well as the basic control layer with the pitch- and elevation-controllers.

2.2 Discretizing the model

We wish to discretize the model found in eq. (4) using the forward Euler method to get the discrete-time state-space model as seen in eq. (5). The method of discretization can be seen in eq. (6). Combining this with eq. (2)

gives us eq. (7), where A and B are the discrete-time system matrices. The discrete time system matrices are shown in eq. (8)

$$x_{k+1} = Ax_k + Bu_k \tag{5}$$

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T_s} \tag{6}$$

$$x_{k+1} = (T_s A_c + I)x_k + (B_c T_s)x_k = Ax_k + Bu_k \tag{7}$$

$$\mathbf{A} = \begin{bmatrix} 1 & T_s & 0 & 0 \\ 0 & 1 & -T_s K_2 & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & -T_s K_1 K_{pp} & 1 - T_s K_1 K_{pd} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ T_s K_1 K_{pp} \end{bmatrix} \tag{8}$$

2.3 Calculating optimal trajectory

2.3.1 Method

In this section we want to find and optimal trajectory that moves the helicopter from x_0 to x_f, shown in eq. (9), and with $\lambda_0 = \pi$ and $\lambda_f = 0$. We assume the elevation angle to be constant in this assignment, and that we can control it independently of pitch with the elevation controller in the basic control layer. To limit the pitch angle we put a constraint on the pitch state at every time-step, k, as seen in eq. (10).

$$x_0 = \begin{bmatrix} \lambda_0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad x_f = \begin{bmatrix} \lambda_f \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{9}$$

$$|p_k| \leq \frac{30\pi}{180}, \quad k \in \{1, ..., N\} \tag{10}$$

The way we construct our optimal trajectory is to formulate our problem as a finite-horizon optimization problem. The objective function of this problem can be expressed as eq. (11). Where we use a time-step of $T_s = 0.25s$ and a horizon of $N = 100$.

$$\phi = \sum_{i=1}^{N} (\lambda_i - \lambda_f)^2 + q p_c i^2 \tag{11}$$

Now we have what we need to construct a QP optimization problem that will allow our helicopter to move along a desired trajectory. Equation (12) is adapted from "Merging Optimization and Control"[1], and fits our purpose

nicely. However we see that the objective function in eq. (12a) and eq. (11) are different so we need to transform eq. (11) to an objective that encompas all states not just λ. Note, that our original objective does not have a linear term, c_t, so all we need is to find a suitable Q_t and R_t. Our transformed Q and R are also time-invariant, and shown in eq. (13).

We also see that the QP expects some constraints on states in eq. (12c) that we do not have in our original problem. This is easily solved by setting the constraints the otherwise unconstrained states to $\pm\infty$.

Lastly we need to incorporate the dynamics of our helicopter into the optimization problem, and this is easily done by letting eq. (12b) equal the discrete-time state-space equations found in section 2.2.

We see that our problem is convex since the objective is convex, as long as $\mathbf{R} \succeq 0$, and the constraints are all linear. Since this is a convex problem it can be solved with any of the algorithms presented in "Numerical Optimization"[2] such as interior-point method or active set method.

$$\min_{z \in \mathbb{R}^n} \phi(z) = \sum_{i=0}^{N} \frac{1}{2} x_{t+1}^\top Q_{t+1} x_{t+1} + c_{t+1} x_{t+1} + \frac{1}{2} u_t^\top R_t u_t \tag{12a}$$

subject to

$$x_{t+1} = Ax_t + Bu_t \tag{12b}$$

$$x^{low} \leq x_t \leq x^{high} \tag{12c}$$

$$u^{low} \leq u_t \leq u^{high} \tag{12d}$$

where

$$z = (x_1^\top, ..., x_N^\top, u_0^\top, ..., u_{N-1}^\top)^\top \tag{12e}$$

$$n = N(n_x + n_u) \tag{12f}$$

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{R} = q \tag{13}$$

We wish to solve our QP, eq. (12), using MATLAB and the built-in function "`quadprog`". In order to do that we need to transform our QP to standard form. This involves altering the objective function to be on the form of eq. (14), where G is a block diagonal matrix.

$$\min_{z \in \mathbb{R}^n} \phi(z) = \frac{1}{2} z^\top G z \tag{14a}$$

where

$$G = \begin{bmatrix} Q_1 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & Q_N & \ddots & & \vdots \\ \vdots & & \ddots & R_0 & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & R_{N-1} \end{bmatrix} \tag{14b}$$

We can also extend our linear model constraints by realizing that every state in eq. (12b) depends upon the previous state, to give us equality constraints with dimension z. The derivation of this method is explained in[1], and the result is summarized in eq. (15). Extending the state and input constraints to hold for z instead of x_t and u_t is quite trivial.

$$\underbrace{\begin{bmatrix} I & 0 & \cdots & 0 & -B & 0 & 0 & 0 \\ -A & \ddots & 0 & 0 & 0 & \ddots & 0 & 0 \\ 0 & \ddots & \ddots & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & -A & I & 0 & 0 & 0 & -B \end{bmatrix}}_{A_{eq}} \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_N \\ u_0 \\ \vdots \\ u_N \end{bmatrix}}_{z} = \underbrace{\begin{bmatrix} Ax_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{B_{eq}} \tag{15}$$

2.3.2 Implementation in MATLAB

Our realization of the optimization problem in the above section can be found in the appendix, appendix B.2. Note that we use a couple functions that are not included in this report, they are: `gen_constraints`, `gen_q` and `gen_aeq`. They were handed out along with the project. They simply aid in converting the QP from the form eq. (12) to eq. (14-15).

When the problem is on our desired form we use the MATLAB-function `quadprog` to minimize our objective function, subject to its constraints and generate an optimal input u^*. The MATLAB code used to solve this problem is found in appendix B.2.

2.3.3 Evaluation of results

Here we will investigate the impact of changing q in eq. (12a). Changing q corresponds to changing R in eq. (13), which again equals altering `P1` in the code in appendix B.2. The difference in optimal trajectory on all four

states, x, as well as the setpoint for our pitch-controller, $u = p_c$, can be seen as the magenta curve in fig. 2, fig. 3 and fig. 4 in appendix A. Henceworth, the plots and code will match in variable name, while our equations coincide with the assignment text.

The parameter q can be considered a cost on our optimization output, p_{ci}. At every time step we wish to use as little p_{ci} as possible, while still arriving at the desired λ_f as soon as possible. We can think of an increase in q as a stricter restriction on the pitch angle the helicopter can maintain over time. This is seen in the subplots of u, as q increases the time that p_c is equal its constraint, eq. (10), decreases.

Also note that increasing q makes the difference between λ_i and λ_f less relevant. With a high q, there is less importance on reaching the desired destination quickly, as the solution to the optimization problem will be to keep p_{ci} lower at the cost of maintaining a larger difference between desired and actual travel for a longer time.

Note that there are some problems with our objective function eq. (12a), particularly the term $\lambda_i - \lambda_f$. Imagine a scenario where the helicopter approaches the desired point λ_f with a fast travel rate. At the instant it reaches the desired point, the optimal input would be to set the pitch angle to zero according to the objective function. This causes the helicopter to glide past the desired end-point and counter measures must be made to return to the right λ_f. This could cause the helicopter to oscilate around the desired destination.

2.4 Implementing the optimal control sequence in Simulink

2.4.1 Implementation

The optimal trajectory found in the section above is to be implemented as a sequence of set-points to our pitch-controller. The aim here is to get the physical plant to behave as expected, namely that the helicopter flies half a rotation around its λ-axis at neutral elevation.

However, the sensors on our helicopter are relative, and initialized at zero-values every time the helicopter is run. Meaning that the physical helicopter-setup plant has a zero elevation angle when it is resting on the table, while our model has a zero elevation angle when the helicopter is paralell to the surface. Our model also operate under the assumption that our initial travel is half a rotation, $\lambda_0 = \pi$, while the sensor-value is zero. We solve this mismatch by adding a -30° offset on elevation-angle, and a 180° offset on travel-angle in our Simulink model.

The optimal trajectory is only valid for $N \cdot T_s = 25s$ and expects the helicopter to be at its initial position of x_0 when the sequence starts. To get the helicopter to the desired starting point we pad the input-sequence with five seconds worth of x_0 states at the start, and to avoid it dropping to the

ground after the optimal sequence is finished we pad with five seconds of x_f states at the end. This optimal input vector is then imported into Simulink with the "`From Workspace`"-block and used as the set-point, p_c, for our pitch controller. The resulting Simulink model can be found in fig. 14.

2.4.2 Results

We collected data from three different executions of the optimal control sequence, each with different values for q. These measurements are plotted together with their corresponding optimal control sequence in the figures (2-4) in appendix A.

Note that the tracking between the predicted state from the optimization problem and measured state are relatively good for the states p and $\dot{p}$ no matter the value for q. However for the state we try to control and its derivative, λ and r, the situation is much worse. We are never able to reach our desired end-state x_f.

The optimal control sequence implemented above is an open-loop dynamic optimization, meaning that any modeling error, no matter how small will lead to deviations from from the optimal trajectory, as we have observed. It's apparent that there's more inertia in the physical system than in the model. The pitch required to move the helicopter 180° is underestimated in the calculated trajectory compared to the actual plant, and the pitch required to stop it at x_f is put in prematurely. This makes the helicopter change travel direction before it reaches it's destination and it drifts back towards and past the position. The lack of feedback in our method means that we are not able to correct for these model errors. We will see how we can compensate for this by introducing feedback in section 3.

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

3.1 Introducing Feedback

We wish to introduce feedback into our optimal controller. By solving the finite-horizon optimization problem presented in section 2 we get an optimal input sequence, u^*, and an optimal trajectory, x^*. We incorporate feedback by introducing the manipulated variable in eq. (16). With this choice of feedback the optimal input, u_k^* will be implemented as long as the helicopter follows the optimal trajectory, x_k^*, if not the feedback-term will kick into action. We need to decide on a suitable choice of the feedback matrix, K, and we chose to implement it as a LQ controller that minimizes the optimization problem in eq. (17). $Q_{fb} = I$ and $R_{fb} = 1$ serve the purpose of weighting matrices that state how much we penalize deviations in state, and how much we penalize use of the manipulated variable respectively. As with the objective function weighting matrices we will refer to the LQR feedback matrices as Q_{fb} and R_{fb} in the report, while in the code they are the variables $Q2$ and R.

$$u_k = u_k^* - K^\top (x_k - x_k^*) \tag{16}$$

$$J = \sum_{i=0}^{\infty} \Delta x_{i+1}^\top Q_{fb} \Delta x_{i+1} + \Delta u_i^\top R_{fb} \Delta u_i \tag{17a}$$

for a linear model

$$x_{i+1} = Ax_i + Bu_i \tag{17b}$$

where

$$\Delta x = x - x^* \tag{17c}$$

$$\Delta u = u - u^* \tag{17d}$$

To find our matrix K we used the function "`dlqr`" in MATLAB. Our code implementation to find this feedback gain matrix can be seen in appendix B.3 at the bottom under the part called `%Feedback Optimization`.

3.2 Implementing Feedback in Simulink

With our gain matrix, K, in place we can implement the feedback into our Simulink model as seen in fig. 15 in appendix C. Note that the feedback itself is implemented in the subsystem "`Trajectory correction`", the details of which is shown in fig. 16 in appendix C.

The "`Trajectory correction`"-subsystem encapsulates the LQ feedback correction. Here we take in the optimal input and trajectory, u^* and

x^*, from the MATLAB workspace and produce a corrected input satisfying eq. (16) that we use as the set-point, p_c, for our pitch controller.

The results from this feedback introduction is seen in figures (5-7). If we focus on the subplots showing the optimal λ^* against the measured λ we see a great improvement in tracking compared to the open-loop scenario in section 2. As such we can conclude that the introduction of feedback improved our tracking of optimal trajectory considerably. The lowest value for R_{fb} seems to provide the best over all tracking.

Note that more time could be spent adjusting the weighting matrices Q_{fb} and R_{fb} in our feedback, and that would lead to an even better tracking. We were satisfied with our results and didn't spend more time tuning the feedback in this part of the assignment.

3.3 Alternatives to LQ feedback

An alternative to using an LQ controller to provide feedback is to use model predictive control (MPC) instead. An MPC controller start by solving a finite-horizon quadratic optimization problem, but instead of implementing the whole optimal input sequence you only implement the first, wait for a measurement, and then reoptimize your QP with the masurment as a new inital state, implement the new first input, and so on.

By only implementing one input and waiting for a new measurement before the next input is computed we introduce feedback in this control scheme. This means that MPC has built-in feedback and can as such replace both our optimization algorithm and our LQ feedback. To realize MPC with our current setup we would need to get the measurments from QuaRC via Simulink to our MATLAB script at every iteration, this can be accomplished by incorporating our script in the Simulink model as a function block. The optimization function then needs to only provide one input step instead of the whole sequence and wait until it is handed a new state-measurement.

One advantage of MPC is that it couples open loop optimization with feedback so that we do not need to rely on external feedback to correct the deviation of actual state vs. optimal trajectory. This is useful since we can optimize from whatever state the plant is currently in, or estimated to be in, and then optimize from that state towards the desired state.

A disadvantage with MPC is the increased computational complexity as an open loop optimization problem needs to be solved at every time-step. This requires either more efficient algorithms or more computing power, or if this is not available either a lower sample rate for the control loop or a shorter time-horizon to optimize.

By implementing an MPC controller, we would in practice merge the optimization layer and the advanced control layer in figure 8 in the assignment text, as the measured state x would be fed back into the optimization layer aswell.

4 Optimal Control of Pitch/Travel and Elevation with and without Feedback

Now, the task is to calculate an optimal trajectory in two dimensions. The helicopter is moved from x_0 to x_f past an imaginary obstacle causing the elevation angle to change during the flight. We must therefore add the elevation to the state vector; the setpoint e_c is a new manipulated variable in the system.

4.1 Extending the model

So we add the dynamics for the elevation from eq. (1) to the continous system in eq. (2)-(4) in section 1. this gives us a new state-vector x, input u, and system matrices as shown in eq. (18)-(19).

$$x = \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \\ e \\ \dot{e} \end{bmatrix}, \quad u = \begin{bmatrix} p_c \\ e_c \end{bmatrix} \tag{18}$$

$$\begin{aligned} \mathbf{A_c} &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3K_{ep} & -K_3K_{ed} \end{bmatrix}, \\ \mathbf{B_c} &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3K_{ep} \end{bmatrix} \end{aligned} \tag{19}$$

4.2 Discretizing the extended model

Utilizing the forward Euler method (eq. (6)-(7)) we arrive at the discretized system in eq. (20)

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

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ T_s K_1 K_{pp} & 0 \\ 0 & 0 \\ 0 & T_s K_3 K_{ep} \end{bmatrix} \tag{20}$$

4.3 Extending the optimization problem

4.3.1 Implementing the obstacle

Next we introduce the obstacle into the problem, which is modelled by a nonlinear inequality constraint given by eq. (21)

$$c_{obstacle}(x_k) = \alpha \exp(-\beta(\lambda_k - \lambda_t)^2) - e_k \leq 0, \quad \forall k \in \{1, ..., N\}. \tag{21}$$

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

The MATLAB code for generating this constraint can be found in appendix B.5.

4.3.2 Updating the optimization problem

As we have added states and inputs to the model we need to reflect those changes in the objective function.

The optimization problem in eq. (12) now has the objective function stated in eq. (22),

$$\phi = \sum_{i=1}^{N} (\lambda_i - \lambda_f)^2 + q_1 p_{ci}^2 + q_2 e_{ci}^2 \tag{22}$$

with updated constraints to match the new system matrices and the addition of the non-linear inequality constraint $c_{obstacle}$ in eq. (21). We also have a slightly different z as each vector x_i has 2 additional states and the

elements u_i are now vectors instead of scalars. Furthermore this objective function requires extended $\mathbf{Q}$ and $\mathbf{R}$ matrices as shown in eq. (23).

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} q_1 & 0 \\ 0 & q_2 \end{bmatrix} \tag{23}$$

4.3.3 Calculating optimal trajectory

Because we now have introduced a non-linear constraint to our optimization problem, we can no longer solve it using the MATLAB function `quadprog` so we're going to use `fmincon` instead. This function expects us to parse an objective function and a function for the non-linear constraints as arguments, and the implementation can be found in appendix B.4 and B.5. Because of the increased complexity of the problem we reduced the horizon to $N = 40$, and started with values $q_1 = q_2 = 1$.

The resulting trajectory for the elevation can be found in fig. 12 in appendix A.

4.4 Implementing trajectory in the plant with and without feedback

The Simulink model used in this section can be found in fig. 17 and fig. 18 in appendix C. Running the helicopter with and without feedback using the optimal input calculated by `fmincon` gave slightly different results. The biggest differences can naturally be seen on the travel, λ, and elevation, e. The result can be found in fig. 8 and fig. 9 in appendix A. Both the responses are without tuning of the matrices Q and R. With feedback the feedback matrices used for generating K are shown in eq. (24)

$$\mathbf{Q_{fb}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R_{fb}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{24}$$

We see from the results that without feedback we still have the problem where we never reach our desired end state x_f, as the helicopter misses the travel destination λ_f completely. With feedback we are able to reach our destination but neither running the helicopter with nor without feedback gets even close of clearing the obstacle.

After tuning the model with feedback we ended up with $q_1 = 2,\ q_2 = 1$, and the feedback matrices in eq. (25).

$$\mathbf{Q_{fb}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}, \quad \mathbf{R_{fb}} = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \tag{25}$$

This led to the result found in fig. 10 and fig. 11 in appendix A. We see that the helicopter is still not able to clear the obstacle, but it comes a lot closer. Why it isn't able to fly over the obstacle is further discussed in section 4.5. We started the tuning process by trying to decrease the error $e_c - e$. we raised the corresponding weight in Q_{fb} to make it prioritize minimizing this error more. We kept raising it until we had a satisfyingly fast response in reaching the setpoint without oscillating too much. We were still too short of clearing the obstacle as it was too aggressive on pitch (discussed further in section 4.5) so we lowered the cost of the input e_c.

4.5 Discussing the decoupling of states

With the model we are using there is no connection between the four first states and the last two. This is of course a gross simplification compared to real life as it doesn't reflect that the vertical forces are diminished by having $p \neq 0$. This combined with the same effect when $e \neq 0$ is what causes the big difference in the calculated trajectory and the actual, measured trajectory for elevation. Reducing the effect of the pitch can be done by putting a more aggressive constraint on it, but this will make correction in λ much slower. Another solution is to use a more realistic starting point for modeling the dynamics of our helicopter that has a higher degree of coupling between pitch and elevation.

To push the elevation closer to clearing the obstacle we can also address the effect of the elevation angle. We assume a linear model close to the equilibrium $\lambda = p = e = 0$, which is a reasonable assumption, but the helicopter doesn't have to deviate far in the elevation angle for the dynamics to be far from linear. Again, one option is to use a more coupled model. Another solution is to extend the "arm" the helicopter is connected to, but that would cause problems with the available space of the facilities we are using. Adding a joint on the arm in attempt to hold the helicopter head level when changing the elevation angle is another solution, but that either requires a sophisticated mechanical solution or will result in the head having very low stability.

A Plots

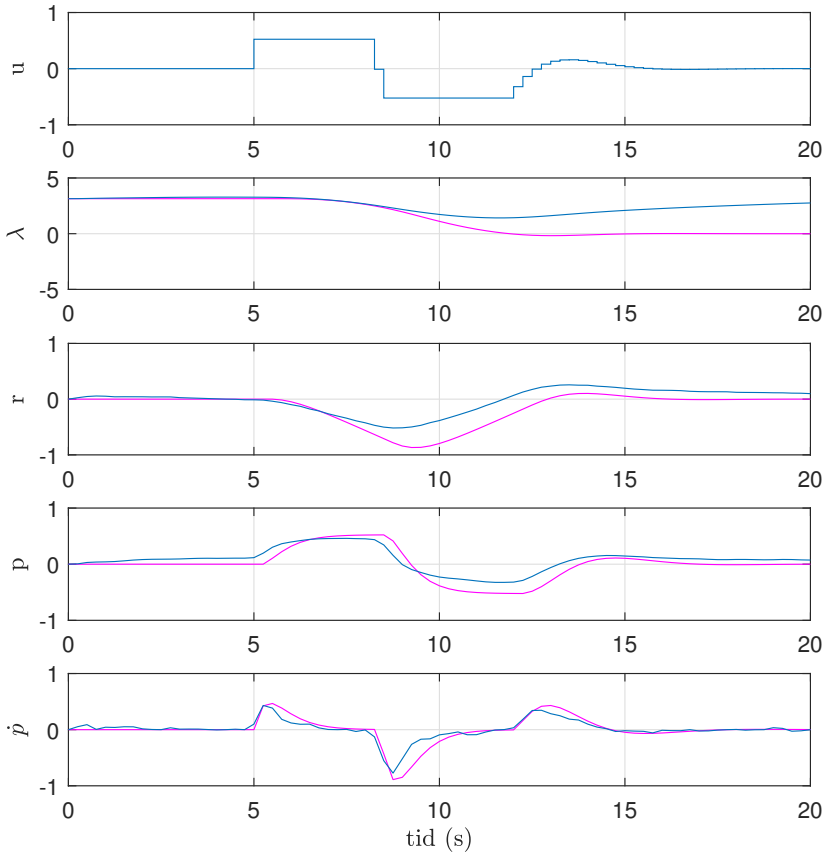


Figure 2: Optimal trajectory in magenta and measured states in blue, no feedback and $P_1 = 0.1$.

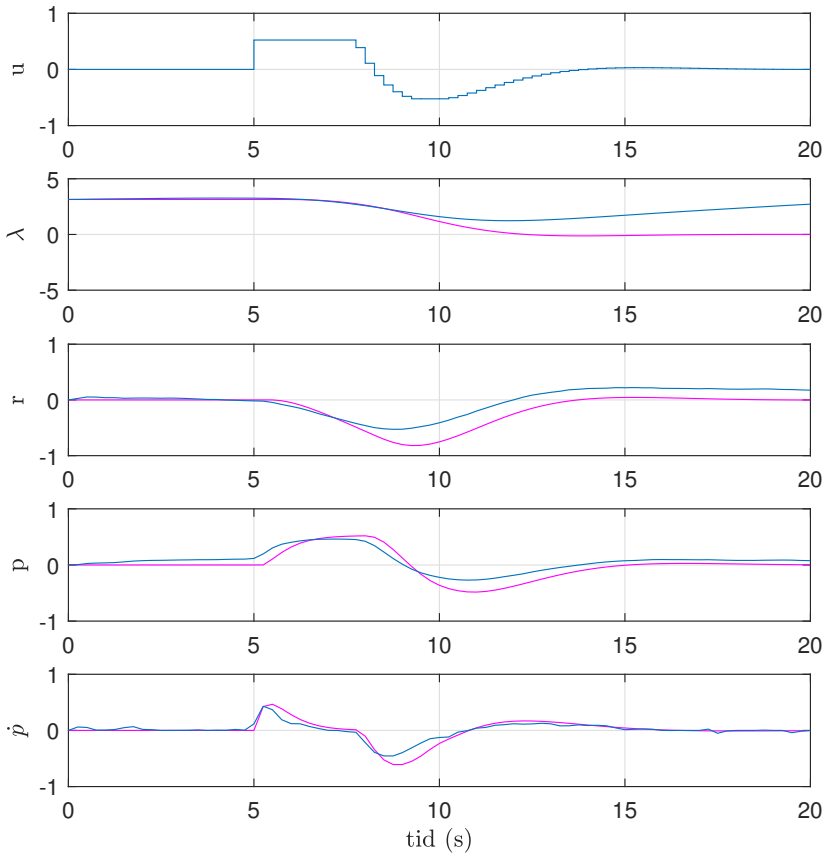


Figure 3: Optimal trajectory in magenta and measured states in blue, no feedback and $P_1 = 1$.

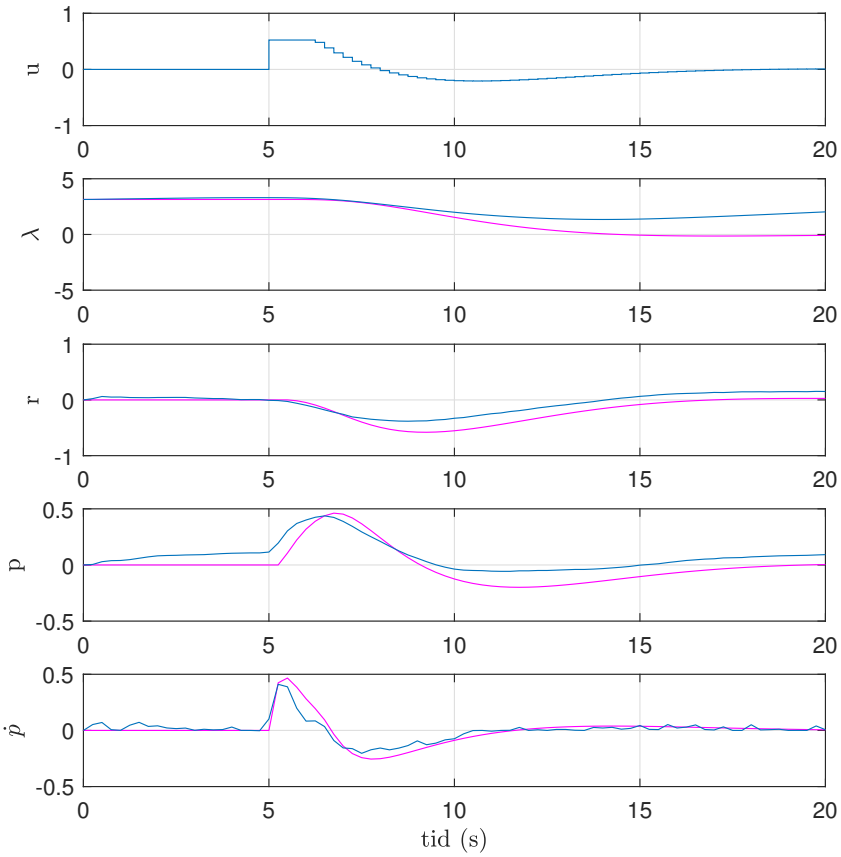


Figure 4: Optimal trajectory in magenta and measured states in blue, no feedback and $P_1 = 10$.

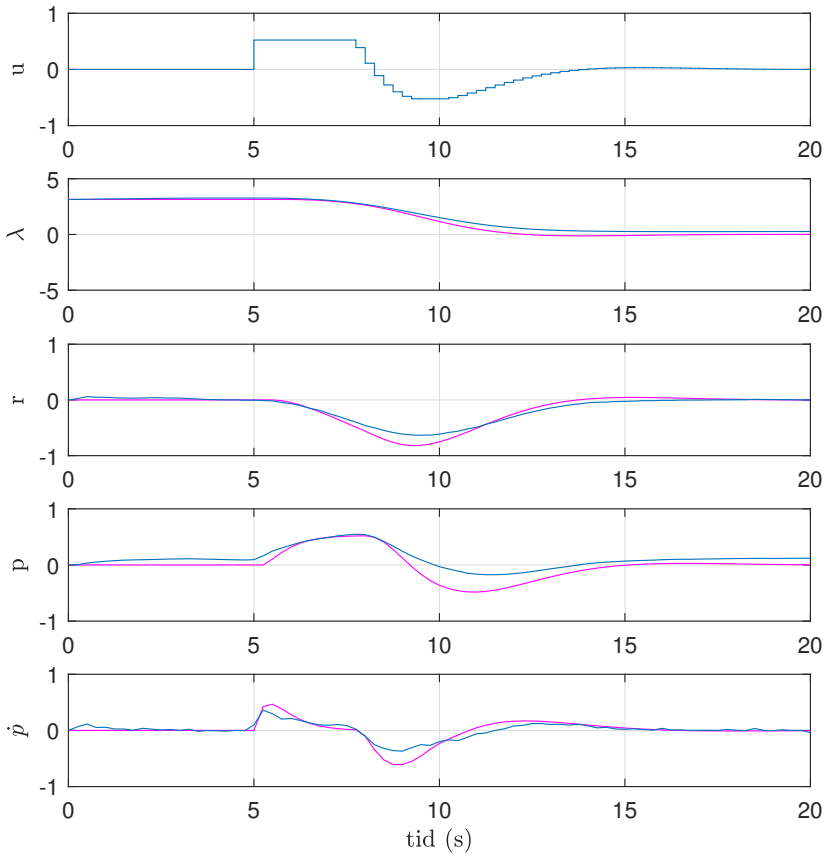


Figure 5: Optimal trajectory in magenta and measured states in blue, with feedback and $P_1 = 1$ and $R = 0.1$.

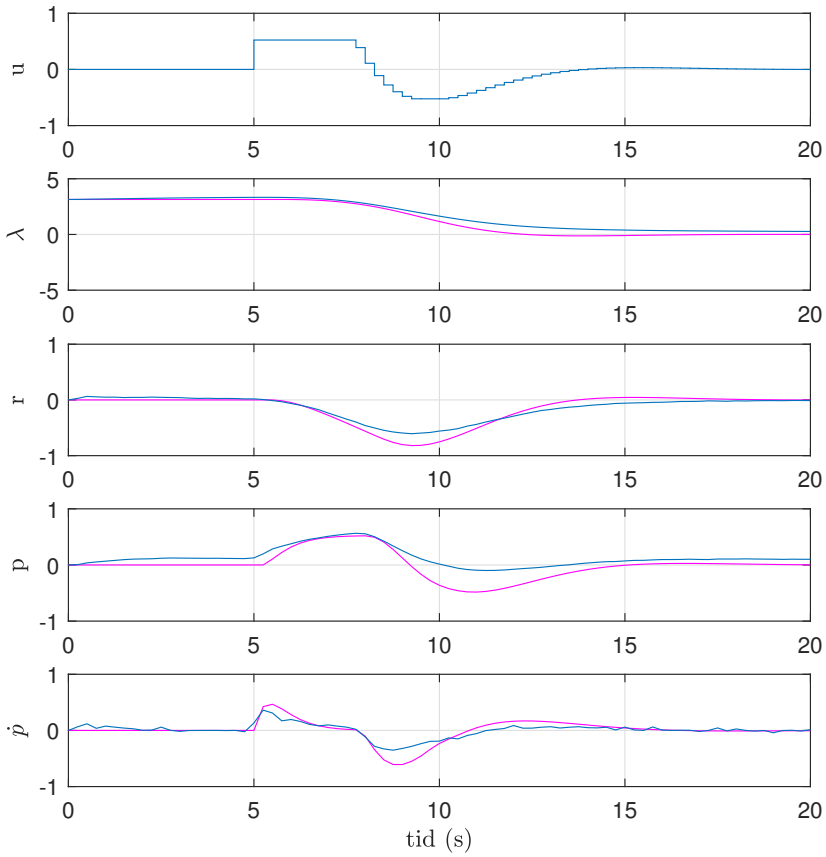


Figure 6: Optimal trajectory in magenta and measured states in blue, with feedback and $P_1 = 1$ and $R = 1$.

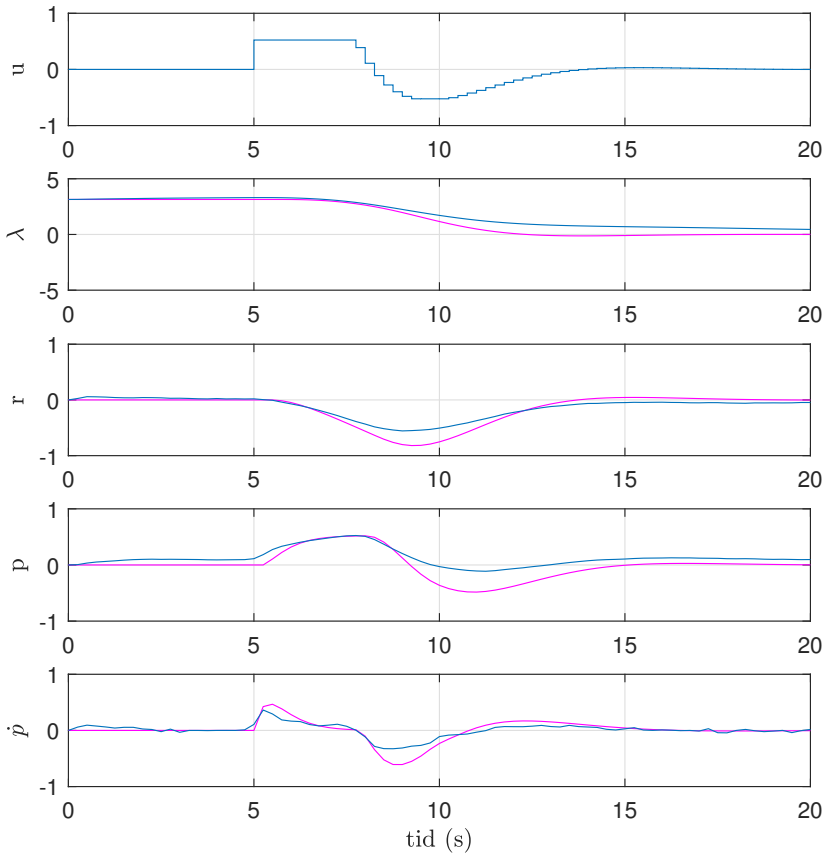


Figure 7: Optimal trajectory in magenta and measured states in blue, with feedback and $P_1 = 1$ and $R = 10$.

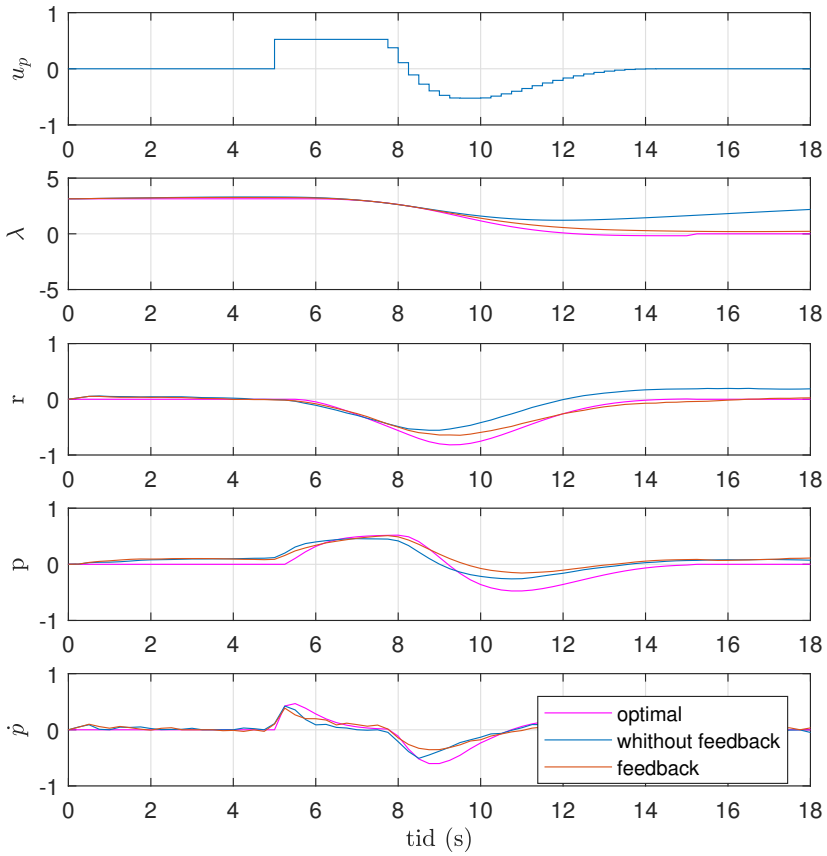


Figure 8: Comparison of feedback and no feedback for the extended system with untuned parameters. Showing the pitch-dependent states.

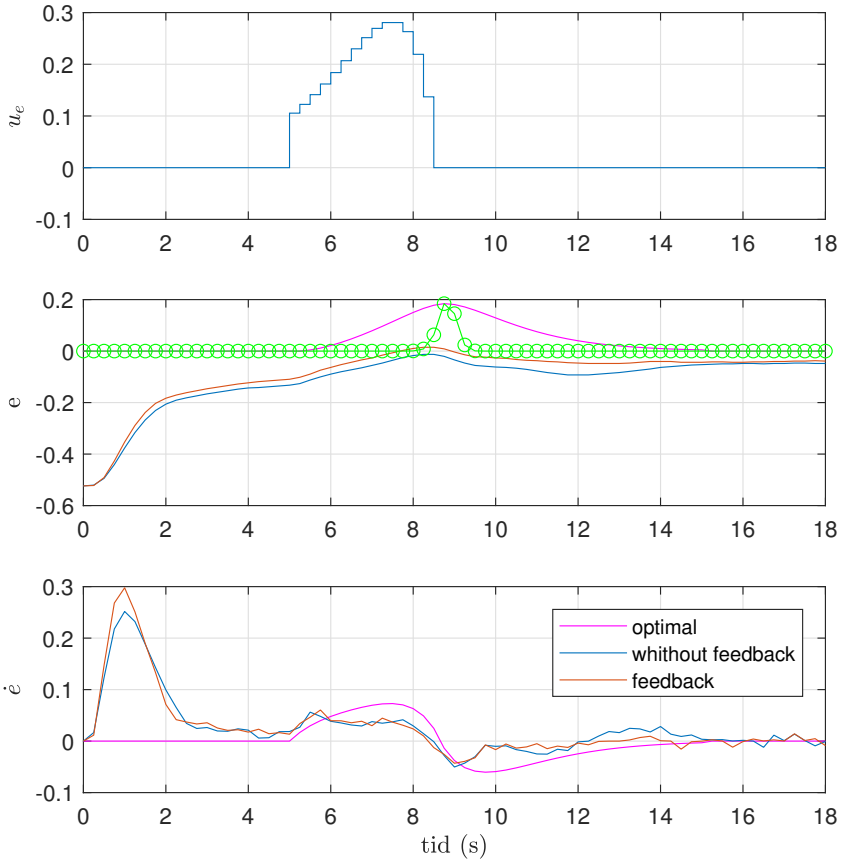


Figure 9: Comparison of feedback and no feedback for the extended system with untuned parameters. Showing the elevation-dependent states. Obstacle plotted in green.

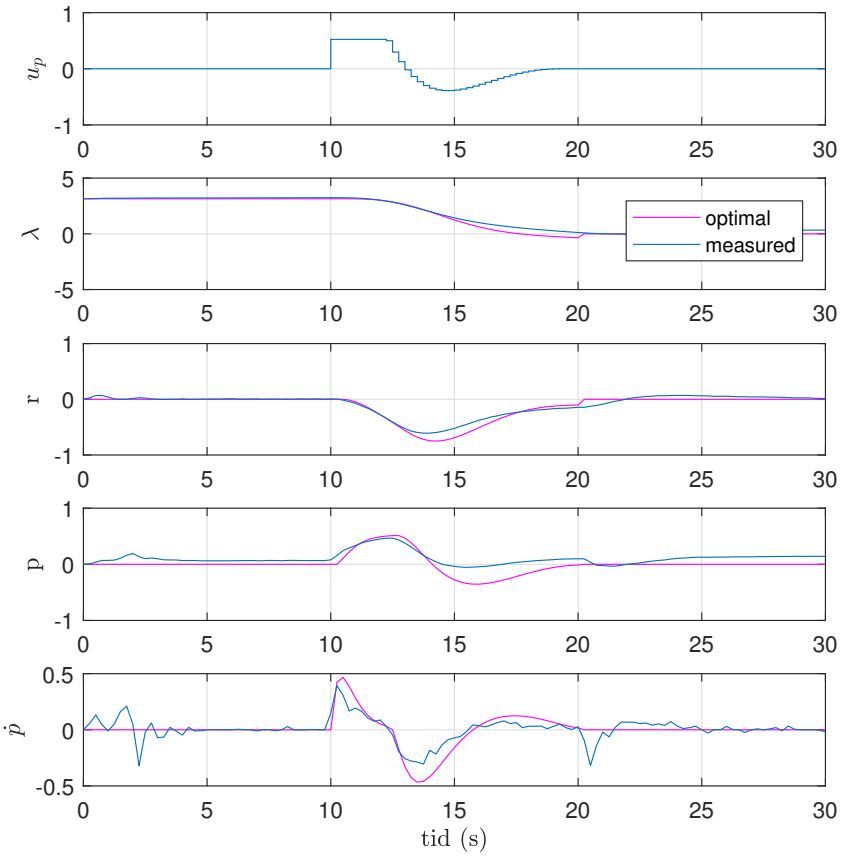


Figure 10: Tuned response of the helicopter compared to the optimal trajectory. Showing pitch-dependant states.

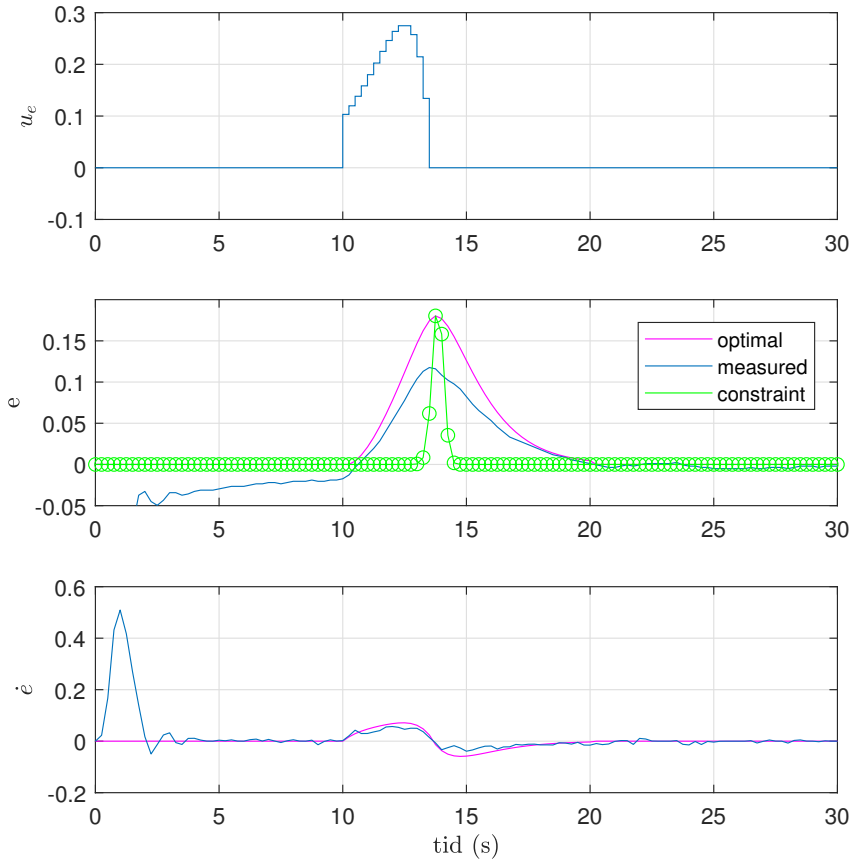


Figure 11: Tuned response of the helicopter compared to the optimal trajectory. Showing elevation-dependant states. Obstacle plotted in green.

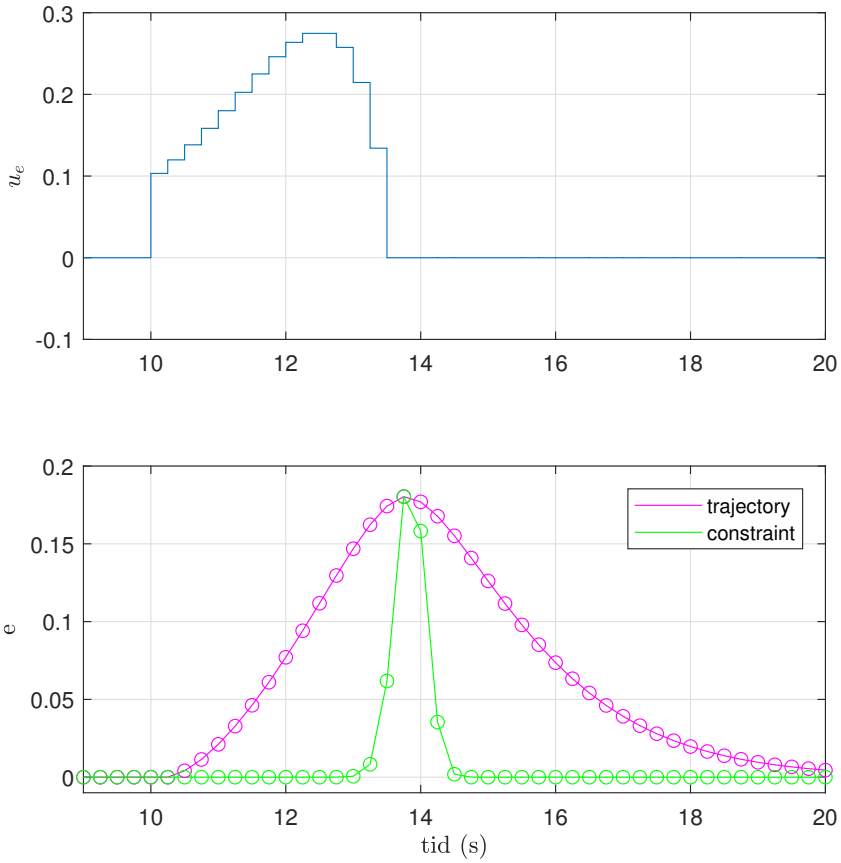


Figure 12: Plot of optimal trajectory over the obstacle constraint.

B MATLAB Code

B.1 init.m

```
clear all;
close all;
clc;

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

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

% Identified voltage sum and difference
V_s_eq = 4.7; % Identified equilibrium voltage sum.
V_d_eq = 0.0; % Identified equilibrium voltage difference.

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

B.2 problem_2.m

```
%Initialization and model definition

clc;
clear;

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  -delta_t*K_1*K_pp   1 - delta_t*K_1*K_pd];

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      = -(30*pi/180);                     % Lower bound on control
uu      = (30*pi/180);                     % Upper bound on control

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

% Generate constraints on measurements and inputs
[vlb,vub]       = gen_constraints(N,M,xl,xu,ul,uu); % 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 = 10;                                  % Weight on input
Q = gen_q(Q1,P1,N,M);                     % Generate Q, hint: gen_q
```

```
c = zeros(N*mx+M*mu,1)';% 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(N*mx,1);                      % Generate b
beq(1:4) = A1*x0;


%Solve QP problem with linear model
tic
[z,lambda] = quadprog(Q,c,[],[],Aeq,beq,vlb,vub,z0);
t1=toc;

% Calculate objective value
phi1 = 0.0;
PhiOut = zeros(N*mx+M*mu,1);
for i=1:N*mx+M*mu
  phi1=phi1+Q(i,i)*z(i)*z(i);
  PhiOut(i) = phi1;
end

%Extract control inputs and states
u  = [z(N*mx+1:N*mx+M*mu);z(N*mx+M*mu)]; % Control input from solution

x1 = [x0(1);z(1:mx:N*mx)];              % State x1 from solution
x2 = [x0(2);z(2:mx:N*mx)];              % State x2 from solution
x3 = [x0(3);z(3:mx:N*mx)];              % State x3 from solution
x4 = [x0(4);z(4:mx:N*mx)];              % State x4 from solution

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

u   = [zero_padding; u; zero_padding];
x1  = [pi*unit_padding; x1; zero_padding];
x2  = [zero_padding; x2; zero_padding];
x3  = [zero_padding; x3; zero_padding];
x4  = [zero_padding; x4; zero_padding];


%Generating optimal input for the simulation
optimal_input = timeseries(u, t);
```

B.3 problem_3.m

```
%Initialization and model definition

clc;
%clear;

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  -delta_t*K_1*K_pp   1 - delta_t*K_1*K_pd];
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      = -(30*pi/180);                     % Lower bound on control
uu      = (30*pi/180);                    % Upper bound on control

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

% Generate constraints on measurements and inputs
```

```
[vlb,vub]       = gen_constraints(N,M,xl,xu,ul,uu); % 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 proble
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 = 1;                                 % Weight on input
Q = gen_q(Q1,P1,N,M);                   % Generate Q, hint: gen_q
c = zeros(N*mx+M*mu,1)';% 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(N*mx,1);                      % Generate b
beq(1:4) = A1*x0;

%Solve QP problem with linear model
tic
[z,lambda] = quadprog(Q,c,[],[],Aeq,beq,vlb,vub,z0);
t1=toc;

% Calculate objective value
phi1 = 0.0;
PhiOut = zeros(N*mx+M*mu,1);
for i=1:N*mx+M*mu
  phi1=phi1+Q(i,i)*z(i)*z(i);
  PhiOut(i) = phi1;
end

%Extract control inputs and states
u  = [z(N*mx+1:N*mx+M*mu);z(N*mx+M*mu)]; % Control input from solution

x1 = [x0(1);z(1:mx:N*mx)];              % State x1 from solution
x2 = [x0(2);z(2:mx:N*mx)];              % State x2 from solution
x3 = [x0(3);z(3:mx:N*mx)];              % State x3 from solution
x4 = [x0(4);z(4:mx:N*mx)];              % State x4 from solution

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

```
u   = [zero_padding; u; zero_padding];
x1  = [pi*unit_padding; x1; zero_padding];
x2  = [zero_padding; x2; zero_padding];
x3  = [zero_padding; x3; zero_padding];
x4  = [zero_padding; x4; zero_padding];


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

%Generating optimal input for the simulation
x_star = [x1,x2,x3,x4];
optimal_input = timeseries(u, t);
optimal_trajectory = timeseries(x_star, t);

%Feedback Optimization

\begin{verbatim}
Q2 = zeros(mx,mx);
Q2(1,1) = 1;                            % Weight on state x1
Q2(2,2) = 1;                            % Weight on state x2
Q2(3,3) = 1;                            % Weight on state x3
Q2(4,4) = 1;                            % Weight on state x4
R = 0.1;                                % Weight on input

[K,S,e] = dlqr(A1,B1,Q2,R);
```

B.4 problem_4.m

```
%Initialization and model definition

clc;
%clear;

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          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];
B1 = [0   0           0          delta_t*K_1*K_pp     0          0;
```

```
      0  0           0                0              0          delta_t*K_3*K_ep]';
global N mx
% 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:mx) = x0;

% Bounds
ul      = [-(30*pi/180) -inf]';                  % Lower bound on control
uu      = [(30*pi/180) inf]';                   % Upper bound on control

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

% Generate constraints on measurements and inputs
[vlb,vub]       = gen_constraints(N,M,xl,xu,ul,uu); % 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 = [2 0;
```

```
      0 1];                                    % Weight on input
Q = gen_q(Q1,P1,N,M);                   % Generate Q, hint: gen_q
c = zeros(N*mx+M*mu,1)';% 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(N*mx,1);                     % Generate b
beq(1:6) = A1*x0;

%Solve QP problem with nonlinear model
fun = @(X) (1/2)*X'*Q*X;
options = optimoptions('fmincon','Algorithm','sqp');
tic
z = fmincon(fun,z0,[],[],Aeq,beq,vlb,vub,@gen_nonlcon, options);
t1=toc;

% Calculate objective value
phi1 = 0.0;
PhiOut = zeros(N*mx+M*mu,1);
for i=1:N*mx+M*mu
  phi1=phi1+Q(i,i)*z(i)*z(i);
  PhiOut(i) = phi1;
end

%Extract control inputs and states
u_p  = [z(N*mx+1:mu:N*mx+M*mu);z(N*mx+M*mu)]; % Control input from solution
u_e  = [z(N*mx+2:mu:N*mx+M*mu);z(N*mx+M*mu)];

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

num_variables = 5/delta_t;
zero_padding = zeros(num_variables*2,1);
unit_padding  = ones(num_variables*2,1);


u_p   = [zero_padding; u_p; zero_padding];
u_e   = [zero_padding; u_e; zero_padding];
x1  = [pi*unit_padding; x1; zero_padding];
x2  = [zero_padding; x2; zero_padding];
```

```
x3  = [zero_padding; x3; zero_padding];
x4  = [zero_padding; x4; zero_padding];
x5  = [zero_padding; x5; zero_padding];
x6  = [zero_padding; x6; zero_padding];

[obstacle, temp] = gen_nonlcon(z);
obstacle = [0; zero_padding; obstacle; zero_padding] + x5;


t = 0:delta_t:delta_t*(length(u_p)-1);

%Generating optimal input for the simulation
x_star = [x1,x2,x3,x4,x5,x6];
u = [u_p u_e];
optimal_input = timeseries(u, t);
optimal_trajectory = timeseries(x_star, t);

%Feedback Optimization
Q2 = zeros(mx,mx);
Q2(1,1) = 1;                              % Weight on state x1
Q2(2,2) = 1;                              % Weight on state x2
Q2(3,3) = 1;                              % Weight on state x3
Q2(4,4) = 1;                              % Weight on state x4
Q2(5,5) = 20;                              % Weight on state x5
Q2(6,6) = 2;                              % Weight on state x6
R = [1 0;                                % Weight on input
     0 0.5];

[K,S,e] = dlqr(A1,B1,Q2,R);
```

B.5 gen_nonlcon.m

```
function [c,ceq] = gen_nonlcon(z)
global N mx
alp = 0.2;
bet = 20;
lambda_t = 2*pi/3;

c = alp*exp(-bet*(z(1:mx:N*mx)-lambda_t).^2)-z(5:mx:N*mx);
ceq = [];
end
```

C Simulink models

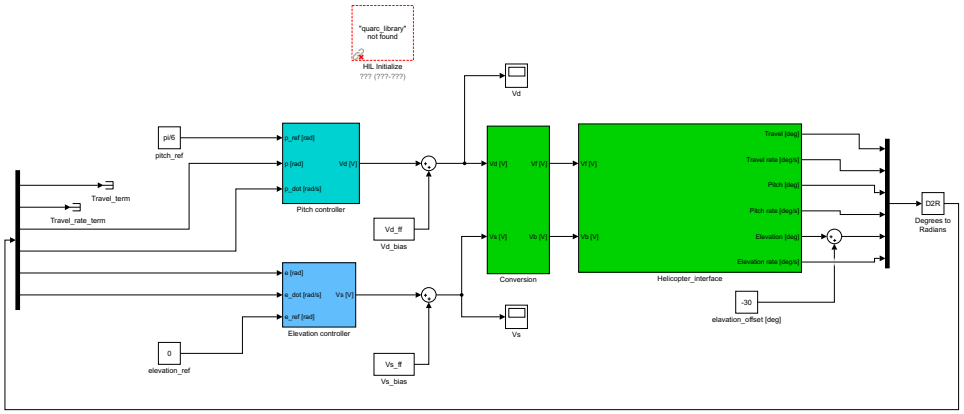


Figure 13: Handed out Simulink model of helicopter with PD pitch controller and PID elevation controller

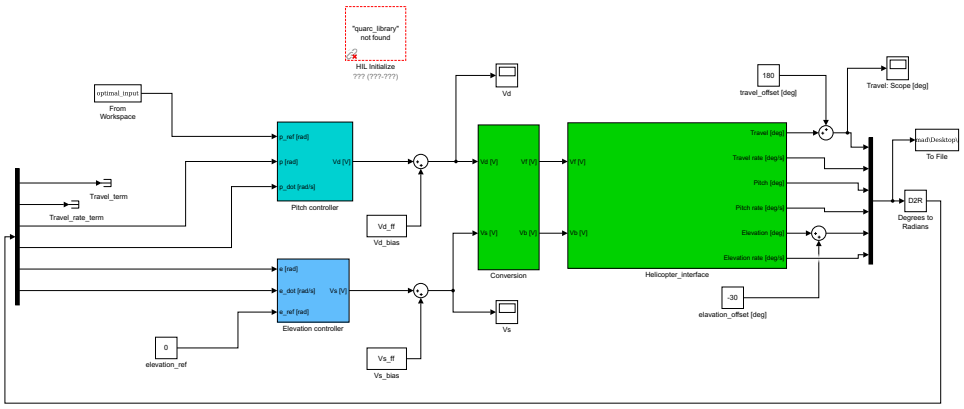


Figure 14: Simulink model with optimization layer on top of basic control layer, no feedback to optimization layer.

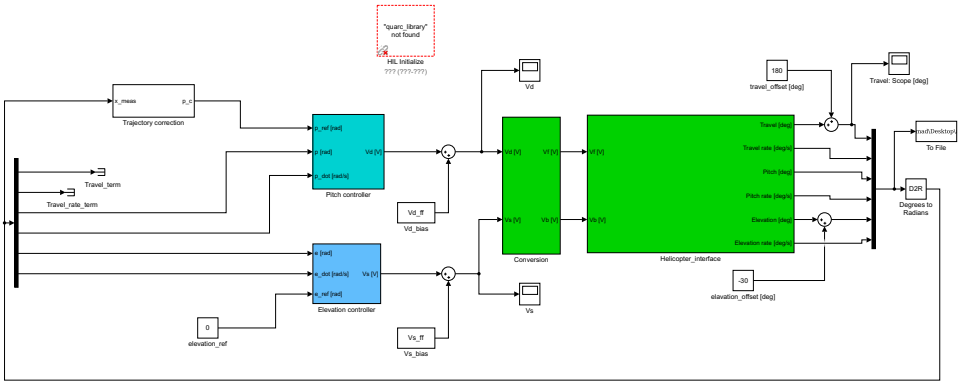


Figure 15: Simulink model with optimization layer on top of basic control layer, this time with feedback to optimization layer in the form of a LQ controller.

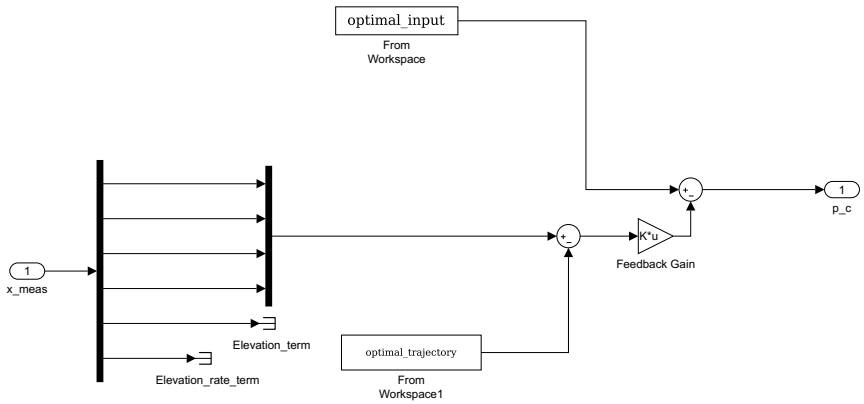


Figure 16: Detailed view of the LQ controller responsible for feedback.

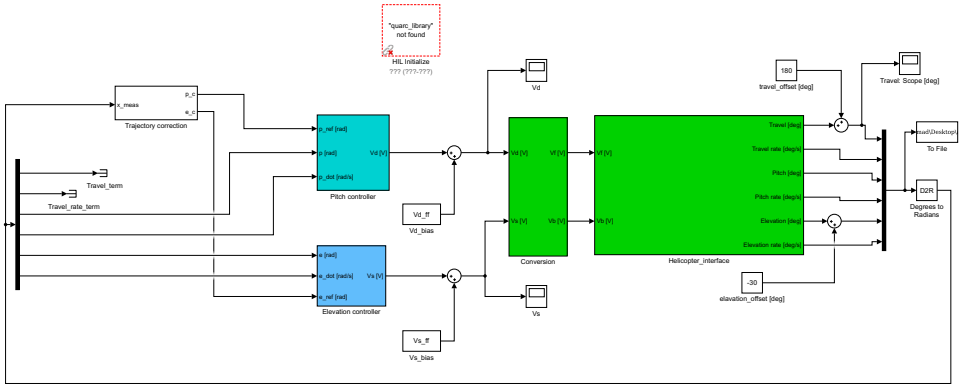


Figure 17: Simulink model with optimization layer on top of basic control layer, this time with feedback to optimization layer in the form of a LQ controller and optimization layer extended to include elevation.

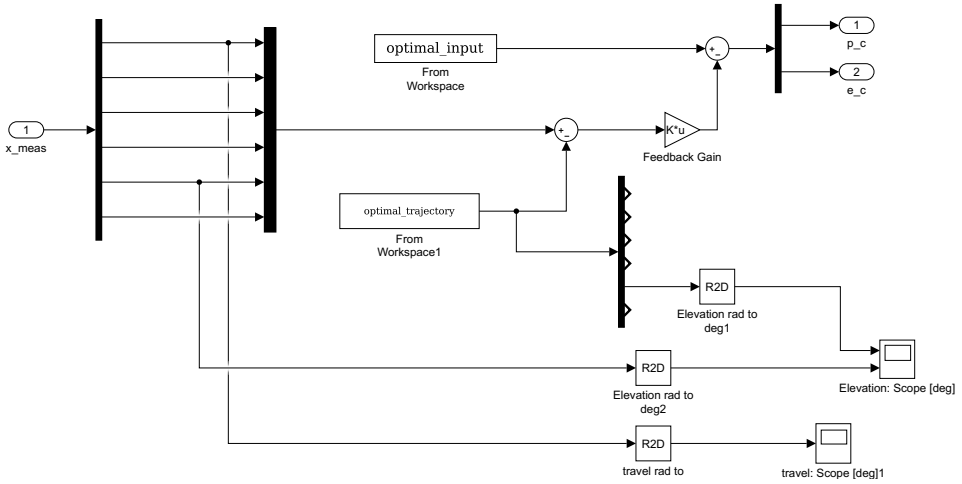


Figure 18: Detailed view of the extended LQ controller responsible for feedback.

References

[1] T.A.N. Heirung B. Foss. *Merging Optimization and Control*. Department of Engineering Cybernetics NTNU, 2016.

[2] S.J. Wright J. Nocedal. *Numerical Optimization*. Springer Science+Business Media, LLC, 2006.