TTK4135 Helicopterlab

768666
768662

March 2018

Department of Engineering Cybernetics

Abstract

This report describes the mandatory lab exercise in the course TTK4135 Optimization and Control. In this course optimization theory is applied to a mathematical model of a helicopter and simulated on a physical model. The optimization and simulation is conducted using computational tools like MATLAB and Simulink. The physical hardware that is exposed to optimization is a Quanser helicopter with three degrees of freedom, travel, elevation and pitch. Optimization of all three degrees of freedom is described in this report. These tools combined with a physical model makes a nice environment to explore the different aspects and methods of optimization.

Contents

1 Introduction

1.1 Exercise overview

During this assignment the work and results of four different exercises will be summarized in this report. All exercises where conducted on helicopter number 1. A set of equations that describes the movement of the helicopter plant were given in the assignment text [1]. By discretizing the model and calculating a optimal trajectory between to given points, we will solve a optimization problem using MATLAB/Simulink with the Optimization Toolbox. As the tasks go on the optimal trajectory will be modeled with an increasing number of states in the state space model. Implementation and results from both with and without the use of feedback is also presented. Overall this report will cover all exercises and tasks in detail, from equations to implementation in MATLAB and finally running the system in Simulink ending with a graphical view of the optimal trajectory versus the actual trajectory.

1.2 Report content

Each exercise is given its own section, while the different tasks in each exercise is included in subsections. Please notice that exercise numbers and the section numbers does not match. References for sections and subsections uses their section- and subsection-numbers respectively, not the exercise numbers. Exercise 1 is an introduction section to get familiar with the lab setup and includes a description of this. In exercise 2 an optimal input that rotates the helicopter 180 degrees is calculated and implemented. The only states of interest in exercise 2 is pitch and travel and no feedback is used. The same number of states, but with the use of feedback is implemented in exercise 3. Elevation is included as the third state in exercise 4 and this exercise is completed with and without feedback. Three appendices is also present, where appendix A includes tables and plots. Simulink diagrams for each exercise is found in appendix B. The last appendix, appendix C, contains MATLAB code for each exercise.

2 Exercise 1: Introduction to Simulink/QuaRC MATLAB

2.1 Task1

This section is meant as an introduction to the helicopter lab setup and to get familiar with how the communications works out from MATLAB code to the physical movement of the hardware. The first thing is to make sure that the lab is functioning as intended. This is done by opening MATLAB and run a given initialization file that is tuned specially for each helicopter. This file sets the parameters of that particularly helicopter and makes the variables available for Simulink. Simulink is a block diagram environment where the model of the helicopter is implemented. This block diagram uses the parameteres that were set in MATLABs workspace by running the initialization file. In order to make Simulink behave differently, the MATLAB script setting the parameteres has to be run first. Simulink, that is running on the computers controlling the helicopter, has a QuaRC (Quanser Real-Time Control) software module added which talks to an I/O card on the computers. To run the helicopter from simulink go to QuaRC $\rightarrow$ Build. This converts the block diagram in Simulink into C-code. The C-code is then run on the module that controls the dc-motors for the helicopter. The helicopter hardware is also equipped with several high-resolution encoders for accurate position feedback. By completing the course TTK4115 Linear System Theory last year this setup was fairly familiar for the authors of this report.

3 Exercise 2: Optimal control of pitch and travel without feedback

3.1 Task 1: Write model on continuous time state space form

The model we will use can be summarized by:

$$\ddot{e} + K_3 K_{ed}\dot{e} + K_3 K_{ep} e = K_3 K_{ep} e_c \tag{1a}$$

$$\ddot{p} + K_1 K_{pd}\dot{p} + K_1 K_{pp} p = K_1 K_{pp} p_c \tag{1b}$$

$$\dot{\lambda} = r \tag{1c}$$

$$\dot{r} = K_2 p \tag{1d}$$

where

$$K_1 = \frac{K_f l_h}{J_p} \qquad K_2 = \frac{K_p l_a}{J_t} \qquad K_3 = \frac{l_a K_f}{J_e}$$

A description of the different parameters and their values is found in Table(1) while the different variables are described in Table (2).

Rearranging (1a) - (1d) and sorting out the differentiated states on the left side gives us:

$$\dot{\lambda} = r \tag{2a}$$

$$\dot{r} = -K_2 p \tag{2b}$$

$$\dot{p} = \dot{p} \tag{2c}$$

$$\ddot{p} = -K_1 K_{pd}\dot{p} - K_1 K_{pp} p - K_1 K_{pp} p_c \tag{2d}$$

$$\dot{e} = \dot{e} \tag{2e}$$

$$\ddot{e} = -K_3 K_{ed}\dot{e} - K_3 K_{K_{ep}} e + K_3 K_{ep} e_c \tag{2f}$$

For now we assume $e = 0$ which results in the given state vector x:

$$\boldsymbol{x} = \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \end{bmatrix} = \begin{bmatrix} \text{Travel} \\ \text{Travel rate} \\ \text{Pitch} \\ \text{Pitch rate} \end{bmatrix} \tag{3}$$

Continuous state space form:

$$\dot{\boldsymbol{x}} = \boldsymbol{A_c x} + \boldsymbol{B_c} u \quad u = p_c \tag{4}$$

$$\begin{bmatrix} \dot{\lambda} \\ \dot{r} \\ \dot{p} \\ \ddot{p} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -K_2 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -K_1 K_{pp} & -K_1 K_{pd} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1 K_{pp} \end{bmatrix} p_c \tag{5}$$

The model shown here models the physical layer and the basic control layer shown in Figure (1). There is no feedback of the states to the optimization layer, but the inner controllers, pitch and elevation, has this type of feedback. This model of the plant includes a model of the system and the two inner controllers of pitch and elevation.

3.2 Task 2: Discretize by using the forward Euler method

The derivative of the state can approximately be written as:

$$\dot{\boldsymbol{x}} \approx \frac{\boldsymbol{x}(k+1) - \boldsymbol{x}(k)}{T_s} \tag{6}$$

By combining equation (4) and (6) we obtain:

$$\frac{\boldsymbol{x}(k+1) - \boldsymbol{x}(k)}{T_s} \approx \boldsymbol{A_c}\boldsymbol{x}(k) + \boldsymbol{B_c}u(k) \tag{7}$$

Rewriting this leads to the discrete approximation:

$$\boldsymbol{x_{k+1}} = (T_s\boldsymbol{A_c} + \mathbf{I})\boldsymbol{x_k} + T_s\boldsymbol{B_c}u_k = \boldsymbol{A_d}\boldsymbol{x_k} + \boldsymbol{B_d}u_k \tag{8}$$

Which is called the forward Euler. By combining the information from equation (5) with the forward Euler we get:

$$\boldsymbol{A_d} = \begin{bmatrix} 1 & T_s & 0 & 0 \\ 0 & 1 & -K_2T_s & 0 \\ 0 & 0 & 1 & T_s \\ 0 & 0 & -K_1K_{pp}T_s & 1 - K_1K_pdT_s \end{bmatrix} \quad \boldsymbol{B_d} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ K_1K_{pp}T_s \end{bmatrix} \tag{9}$$

3.3 Task 3: Calculate an optimal trajectory and input sequence

The optimal trajectory is supposed to go from $x_0 = [\lambda_0 \ 0 \ 0 \ 0]^T$ to $x_f = [\lambda_f \ 0 \ 0 \ 0]^T$, with $\lambda_0 = \pi$ and $\lambda_f = 0$. The objective function is a cost function given as:

$$\phi = \sum_{i=1}^{N} (\lambda_i - \lambda_f)^2 + qp_{ci}^2, \quad q \geq 0 \tag{10}$$

This objective will be minimized within the given constraints:

$$|p_k| \leq \frac{30pi}{180}, \quad k \in 1, ..., N \tag{11}$$

The $(\lambda_i - \lambda_f)^2$ term in (10) ensures a high value of the objective function in the beginning, but it decreases rapidly as the λ_i approaches λ_f at the end of the trajectory. From this it follows that the travel has a lower priority of regulation than the pitch at the end.

With the given endpoint λ_f the objective reduces to:

$$\phi = \sum_{i=1}^{N} \lambda_i^2 + qp_{ci}^2 \tag{12}$$

In order to solve this system with the quadprog function in MATLAB, the system has to be rewritten as a QP problem on standard form:

$$\min_{z} \quad \frac{1}{2}\boldsymbol{z^T H z} \tag{13}$$

subject to

$$\begin{gathered} \boldsymbol{A_{eq} z = B_{eq}} \\ \boldsymbol{lb < z < ub} \end{gathered}$$

To do this the objective function in equation (12) has to be rewritten:

$$\phi = \sum_{i=1}^{N} \begin{bmatrix} \lambda_i \\ r_i \\ p_i \\ \dot{p}_i \end{bmatrix}^T \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_i \\ r_i \\ p_i \\ \dot{p}_i \end{bmatrix} + p_{ci}^T q p_{ci} = \sum_{i=1}^{N} \boldsymbol{x_i^T Q x_i + u_i^T R u_i} \tag{14}$$

which equals the objective function in equation (13) with:

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x_1} \\ \vdots \\ \boldsymbol{x_N} \\ \boldsymbol{u_0} \\ \vdots \\ \boldsymbol{u_{N-1}} \end{bmatrix}, \quad \boldsymbol{H} = 2 \begin{bmatrix} \boldsymbol{Q} & 0 & \cdots & & \cdots & 0 \\ 0 & \ddots & & & & \vdots \\ \vdots & & \boldsymbol{Q} & & & \\ & & & \boldsymbol{R} & & \vdots \\ \vdots & & & & \ddots & 0 \\ 0 & \cdots & & \cdots & 0 & \boldsymbol{R} \end{bmatrix} \tag{15}$$

The state equations can be written as the equality constraint:

$$\boldsymbol{A_{eq} z = B_{eq}} \tag{16}$$

with:

$$\boldsymbol{A_{eq}} = \begin{bmatrix} \boldsymbol{I} & 0 & \cdots & \cdots & 0 & -\boldsymbol{B} & 0 & \cdots & \cdots & 0 \\ -\boldsymbol{A} & \ddots & & & \vdots & 0 & \ddots & & & \vdots \\ 0 & \ddots & & & \vdots & \vdots & & & & \vdots \\ \vdots & & \ddots & \ddots & 0 & \vdots & & & \ddots & 0 \\ 0 & \cdots & 0 & -\boldsymbol{A} & \boldsymbol{I} & 0 & \cdots & \cdots & 0 & -\boldsymbol{B} \end{bmatrix} \tag{17}$$

$$\boldsymbol{B_{eq}} = \begin{bmatrix} \boldsymbol{A x_0} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{18}$$

The lower and upper boundary of **z**:

$$\boldsymbol{lb} < \boldsymbol{z} < \boldsymbol{ub} \tag{19}$$

$$\boldsymbol{lb} = \begin{bmatrix} \boldsymbol{x_{min}} \\ \vdots \\ \boldsymbol{x_{min}} \\ \boldsymbol{u_{min}} \\ \vdots \\ \boldsymbol{u_{min}} \end{bmatrix}, \quad \boldsymbol{ub} = \begin{bmatrix} \boldsymbol{x_{max}} \\ \vdots \\ \boldsymbol{x_{max}} \\ \boldsymbol{u_{max}} \\ \vdots \\ \boldsymbol{u_{max}} \end{bmatrix} \tag{20}$$

The only constraint that is used in this task is the pitch angle constraint, seen in equation (11). The rest of the states are set to $\pm\infty$ to avoid the constraints being active:

$$\boldsymbol{x_{min}} = \begin{bmatrix} -\infty \\ -\infty \\ -\frac{30\pi}{180} \\ -\infty \end{bmatrix}, \quad \boldsymbol{x_{max}} = \begin{bmatrix} \infty \\ \infty \\ \frac{30\pi}{180} \\ \infty \end{bmatrix}$$

$$u_{min} = -\frac{30\pi}{180}, \quad u_{max} = \frac{30\pi}{180}$$

The optimization problem is solved using quadprog with a sampling time of 0.25s and N=100 steps. The optimal trajectory is solved three times, with different values of q. The q values used is 0.1, 1 and 10, and the plots are shown in Figure (2), Figure (3) and Figure (4) respectively. By comparing these plots it is observable that the system is faster and more aggressive with the lower q values. As q increases the system is smoother, but slower. Since q represents the price of using pitch a lower q is expected to give a faster response which complies with the observed result.

3.4 Task 4: Implement the optimal input sequence

The optimal input sequence calculated in subsection (3.3) with $q = 1$ is implemented as input to the helicopter. The calculated optimal trajectory and the actual open loop trajectory is plotted in Figure (5). The plot shows that the measured travel is to high, and it also increases as time passes. This results in a failure to follow the calculated optimal and to meet the optimal endpoint. Since the travelrate is given by pitch the large drift in travel can be explained by the constant deviation in pitch. This deviation in pitch can clearly be seen from one to five seconds of the graph down to the left of Figure (5), where the calculated optimal is zero due to the zero-padding. Zero-padding is added the first five seconds before the optimal input sequence starts and 5 seconds after. This ensures a certain stability of the helicopter before and after the input sequence. The constant deviation in pitch can be explained by the helicopter not lying entirely horizontal on the table when starting, since pitch is given relative to the starting angle. This will cause the computer to think the pitch is 0, but that is not necessarily true in reality.

4 Exercise 3: Optimal Control of Pitch/Travel with Feedback

4.1 Task 1: Introduce feedback in the optimal controller

In this task a feedback on the optimal controller was introduced using the following equation:

$$\boldsymbol{u_k} = \boldsymbol{u_k^*} - \boldsymbol{K^T}(\boldsymbol{x_k} - \boldsymbol{x_k^*}) \tag{21}$$

The gain $\boldsymbol{K}$ was calculated as a LQ controller, and is the gain that minimizes the cost function:

$$J = \sum_{i=0}^{\infty} = \Delta\boldsymbol{x_{i+1}^T}\boldsymbol{Q}\Delta\boldsymbol{x_{i+1}} + \Delta\boldsymbol{u_i^T}\boldsymbol{R}\Delta\boldsymbol{u_i} \tag{22}$$

$\boldsymbol{Q}$ and $\boldsymbol{R}$ are positive diagonal matrices where $\boldsymbol{Q}$ tells how much to penalize state deviation and $\boldsymbol{R}$ tells how much to penalize use of the manipulated input. $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{u}$ are given by:

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

The system was solved using the ”dlqr” function in MATLAB, shown in Figure (24).

4.2 Task 2: Implement feedback on the helicopter

Next the optimal controller in Equation (21) was implemented in Simulink using the $\boldsymbol{K^T}$ value found in the previous subsection and the optimal $\boldsymbol{x}$ and $\boldsymbol{u}$ found in subsection (3.3). The Simulink diagram can be seen in Figure (15) and (16). First the system was simulated with $\boldsymbol{Q}$ and R values as:

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = 1$$

the result can be seen in Figure (6). None of the states followed the optimal trajectory especially good. Travel, plotted under the name lambda, follows the optimal quite badly in the last 12 seconds of the simulation. Travelrate is far from reaching the peak values of the optimal so increasing the penalty of deviation in travelrate could improve travel. In addition, it is desirable to increase the penalty of deviation in travel, to make it deviate less. Having a very high penalty of deviation in travel made travel follow the reference better, however the system control became more aggressive. The Q matrix leading to the best control had quite high penalty for deviation in travel and low penalty for deviation in pitch:

$$\boldsymbol{Q} = \begin{bmatrix} 50 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}, \quad R = 1$$

with the result shown in Figure (7). With these $\boldsymbol{Q}$ and R values, travel became significantly better and aggressive response was only present in a low degree.

4.3 Task 3: MPC controller

An alternative to the feedback in Equation (21) is a model predictive control controller known as MPC controller. In subsection (4.1) and (4.2) the optimal trajectory was calculated once and LQ controller was used in order to keep the system as close to the calculated optimal trajectory as possible. In MPC the optimal trajectory is calculated in the same way followed by the estimation of states and the optimal trajectory recalculation for the next timestep with current state as initial value. Implementing MPC would equal to adding a line from the physical layer to the optimal control layer, as well as removing the advanced control layer and sending the optimal input directly to the basic control layer in Figure (8).

The advantage with MPC is that the calculated optimal trajectory is still the optimal even if the current state deviates from the originally estimated one. However MPC requires much more calculation power which can make it slow and unfit for structures with limited computation power, like microcontrollers. An another advantage with MPC compared to LQR is that LQR has finite horizon calculations, while each timestep for MPC also is calculated in finite horizon, the recalculation for each timestep would basically allow for infinite horizon control.

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

5.1 Task 1: Continuous state space form with two extra states

In this task we need to control the elevation and can therefore no longer assume it to be zero. The new state vector x and input vector u becomes:

$$\boldsymbol{x} = \begin{bmatrix} \lambda \\ r \\ p \\ \dot{p} \\ e \\ \dot{e} \end{bmatrix}, \quad \boldsymbol{u} = \begin{bmatrix} p_c \\ e_c \end{bmatrix} \tag{23}$$

The new continuous state space model, derived from Equation (2a) - (2f), will now become:

$$\dot{\boldsymbol{x}} = \boldsymbol{A_c x} + \boldsymbol{B_c u} \tag{24}$$

with

$$\boldsymbol{A_c} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -K_1K_{pp} & -K_1K_{pd} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -K_3K_{ep} & -K_3K_{ed} \end{bmatrix} \tag{25}$$

$$\boldsymbol{B_c} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp} & 0 \\ 0 & 0 \\ 0 & K_3K_{ep} \end{bmatrix} \tag{26}$$

5.2 Task 2: Discretize the new model using forward Euler method

Using the forward Euler method presented in Equation (8) and the new state space matrices (25) and (26) we obtain the new discrete time state space matrices:

$$\boldsymbol{A_d} = \begin{bmatrix} 1 & T_s & 0 & 0 & 0 & 0 \\ 0 & 1 & -K_2T_s & 0 & 0 & 0 \\ 0 & 0 & 1 & T_s & 0 & 0 \\ 0 & 0 & -K_1K_{pp}T_s & 1-K_1K_pdT_s & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & Ts \\ 0 & 0 & 0 & 0 & -K_3K_{ep}T_s & 1-K_3K_{ed}T_s \end{bmatrix}$$

$$\boldsymbol{B_d} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ K_1K_{pp}T_s & 0 \\ 0 & 0 \\ 0 & K_3K_{ep}T_s \end{bmatrix} \tag{27}$$

5.3 Task 3: Implement inequality constraints on elevation

For this task we have a nonlinear constraint on the elevation, representing a imaginary obstacle, given by:

$$e_k \geq \alpha \text{exp}(-\beta(\lambda_k - \lambda_t)^2) \quad \forall \quad k \in \{1, ..., N\} \tag{28}$$

with constants given as:

$$\alpha = 0.2, \qquad \beta = 20 \qquad \lambda_t = \frac{2\pi}{3}$$

After adding states and inputs the cost function has changed into:

$$\phi = \sum_{i=1}^{N} (\lambda_i - \lambda_f)^2 + q_1 p_{ci}^2 + q_2 e_{ci}^2 \tag{29}$$

Which is equivalent to

$$\sum_{i=1}^{N} \boldsymbol{x_i^T Q x_i} + \boldsymbol{u_i^T R u_i} \tag{30}$$

with

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} q_1 & 0 \\ 0 & q_2 \end{bmatrix} \tag{31}$$

assuming that $\lambda_f = 0$ as in the previous exercises.

In order for matlab to solve the QP it had to be written on standard form as in section (3.3). The hessian $\boldsymbol{H}$ and the expanded state vector $\boldsymbol{z}$ are given by Equation (15), with $\boldsymbol{Q}$ and $\boldsymbol{R}$ from Equation (31). The equality constraints are given by Equation (16) where $\boldsymbol{A_{eq}}$ are given by Equation (17) and $\boldsymbol{B_{eq}}$ by Equation (18) using $\boldsymbol{A}$ and $\boldsymbol{B}$ from Equation (27). We have lower boundary, $\boldsymbol{lb}$, and upper boundary, $\boldsymbol{ub}$, on the state given by Equation (20). Pitch is the only state with constraint, so $\boldsymbol{x_{min}}$, $\boldsymbol{x_{max}}$, $\boldsymbol{u_{min}}$ and $\boldsymbol{u_{max}}$ becomes:

$$\boldsymbol{x_{min}} = \begin{bmatrix} -\infty \\ -\infty \\ -\frac{30\pi}{180} \\ -\infty \\ -\infty \\ -\infty \end{bmatrix}, \quad \boldsymbol{x_{max}} = \begin{bmatrix} \infty \\ \infty \\ \frac{30\pi}{180} \\ \infty \\ \infty \\ \infty \end{bmatrix} \tag{32}$$

$$\boldsymbol{u_{min}} = \begin{bmatrix} -\frac{30\pi}{180} \\ -\infty \end{bmatrix}, \quad \boldsymbol{u_{max}} = \begin{bmatrix} \frac{30\pi}{180} \\ \infty \end{bmatrix} \tag{33}$$

There are no linear inequality constraints but there is the inequality constraint from equation (28) that must be written as:

$$c_1(z) = \alpha \cdot \exp(-\beta(\lambda_k - \lambda_t)^2) - e_k \leq 0 \tag{34}$$

With the system on standard QP form with one nonlinear inequality constraint, the system was simulated for 10 seconds with step size $T_s = 0.25$. The result can be seen in Figure (9).

5.4 Task 4: Implement the optimal input sequence on the helicopter

In this task the helicopter was given the input found in the previous task, in open loop. The response is shown in Figure (10). Travel deviates a lot and gets worse as time passes, just as for the open loop system in section (3.4). Elevation deviates as well but gets better with time. The deviation in elevation can easily be explained by the large difference in initial value between the simulated elevation and the measured one. It is also observable that the elevation has not reached the optimal elevation at the time the zero-padding ends. So the optimal input control sequence is applied even though the elevation is below its expected initial position.

Afterwards the helicopter were given the same input in closed loop using Equation (21) with $\boldsymbol{K^T}$ as the matrix that minimizes (22) when:

$$\boldsymbol{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{35}$$

The implementation in Simulink can be seen in Figure (17) and (18) while the plot can be seen in Figure (11).

It is desired to have a minimum deviation in travel and elevation, and as seen in the closed loop plot there is a large deviation on both. As mentioned is section (3.4) deviation in pitch leads to deviation in travel. There is a constant deviation on pitch from the simulation start, so when it actually is equal to $\boldsymbol{x}^*$ it thinks its deviating. Because of this its desirable to allow deviation in pitch, and therefor the element in $\boldsymbol{Q}$ representing the price of deviation in pitch was reduced. Since elevation struggled to reach its reference the price for deviation in elevation was increased. This made it better, but it was still bad, so the price of using the manipulated input controlling elevation was decreased as well. When the control sequence started, elevation was quite far from the desired height. To fix this as fast as possible there is need for a high value on the derivative of elevation, however the open loop optimal tells the derivative to be zero. Because of this issue, the cost of deviation on the elevations derivative was reduced. After some trying $\boldsymbol{Q}$ and $\boldsymbol{R}$ ended up as:

$$\boldsymbol{Q} = \begin{bmatrix} 10 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.05 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix}, \quad \boldsymbol{R} = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{36}$$

The results of this tuning can be seen in Figure (12). This resulting system had a bit aggressive control, but both elevation and travel followed the optimal trajectory fantastically.

5.5 Task 5: Notice the first 4 states are completely decoupled

In the model derived from Equation (2a) - (2f) elevation and its derivative is completely decoupled from the other states. This is not the case in reality. Elevation is dependant on pitch and travel is dependant on elevation as seen in these equations:

$$\ddot{e} = \frac{L_2}{J_e} \cdot cos(e) + \frac{L_3}{J_e} \cdot V_s cos(p) \tag{37}$$

$$\ddot{\lambda} = \frac{L_4}{J_\lambda} \cdot V_s sin(p) cos(e) \tag{38}$$

which results in Equation (2f) and (2b) if linearized around $p = 0$ and $e = 0$ with a PD regulator for V_s.

A possible solution to improve the calculated optimal trajectory is to use a nonlinear model instead of a linearized one. This would fit the model better, be more precise and would work even if the system moves away from the linearization point.

6 Conclusion

The main objective for this assignment was to make the helicopter follow a calculated optimal trajectory between two points with constraints using an optimal input sequence. Although some tuning were performed, deviations between the optimal- and the measured-trajectory were observed in all exercises, and this is caused by a number of things. The zeroing of the pitch sensor reading at the start of the simulations when the helicopter is not perfectly horizontal causes the travel to drift before the input sequence is initiated. This drift in travel is clearly seen in some of the plots and moves the helicopter away from the desired initial point. When the linearized model moves away from its linearization point, another reason for the observed deviation arises. Some more tuning could have been done to improve the response, but the problem with linearization and drift in travel would still be present. There are two main things that could solve the problem with deviation, and that is the use of a nonlinear model and the use of a MPC controller. The nonlinear model does not have to take the linearization point into account and the MPC would calculate a new optimal trajectory for each timestep , which means that the optimal input would be optimal for were the helicopter is and not where it should have been. One of the drawbacks with a MPC controller is that it is not suited for all systems because it demands some computational power.

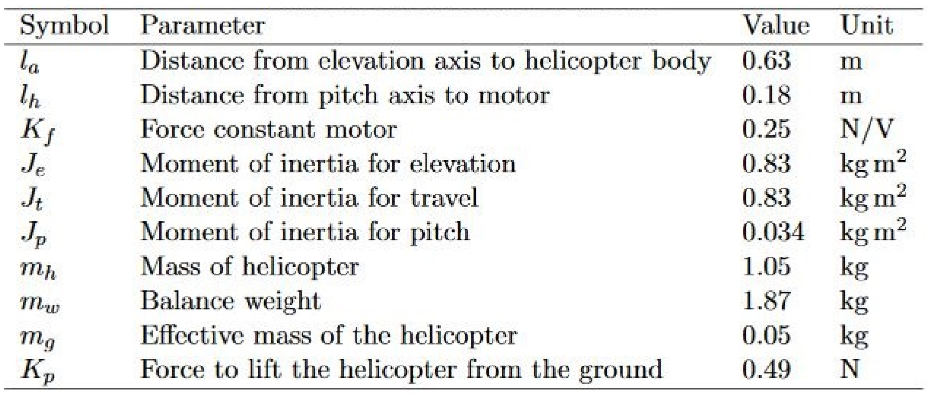
References

[1] NTNU Department of Engineering Cybernetics. *Assignment text.* Trondheim, February, 2018.

[2] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer Science+Business Media, 233 Spring Street, New York, USA, second edition, 2006.

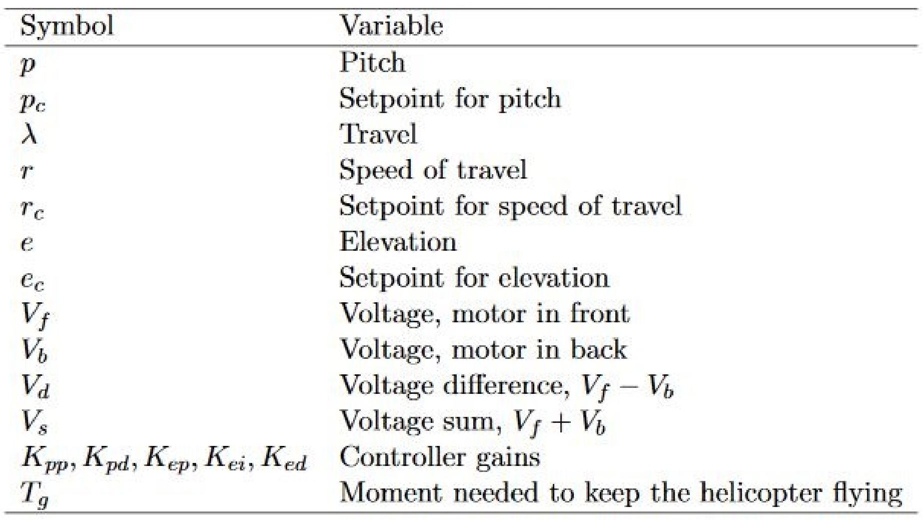
[3] Bjarne Foss and Tor Aksel N. Heirung. *Merging Optimization and Control.* NTNU, Trondheim, 2016.

A Tables and figures



Symbol	Parameter	Value	Unit
l_a	Distance from elevation axis to helicopter body	0.63	m
l_h	Distance from pitch axis to motor	0.18	m
K_f	Force constant motor	0.25	N/V
J_e	Moment of inertia for elevation	0.83	$\mathrm{kg\,m^2}$
J_t	Moment of inertia for travel	0.83	$\mathrm{kg\,m^2}$
J_p	Moment of inertia for pitch	0.034	$\mathrm{kg\,m^2}$
m_h	Mass of helicopter	1.05	kg
m_w	Balance weight	1.87	kg
m_g	Effective mass of the helicopter	0.05	kg
K_p	Force to lift the helicopter from the ground	0.49	N

Table 1: Parameters and Values.



Symbol	Variable
p	Pitch
p_c	Setpoint for pitch
λ	Travel
r	Speed of travel
r_c	Setpoint for speed of travel
e	Elevation
e_c	Setpoint for elevation
V_f	Voltage, motor in front
V_b	Voltage, motor in back
V_d	Voltage difference, $V_f - V_b$
V_s	Voltage sum, $V_f + V_b$
$K_{pp}, K_{pd}, K_{ep}, K_{ei}, K_{ed}$	Controller gains
T_g	Moment needed to keep the helicopter flying

Table 2: Variables.

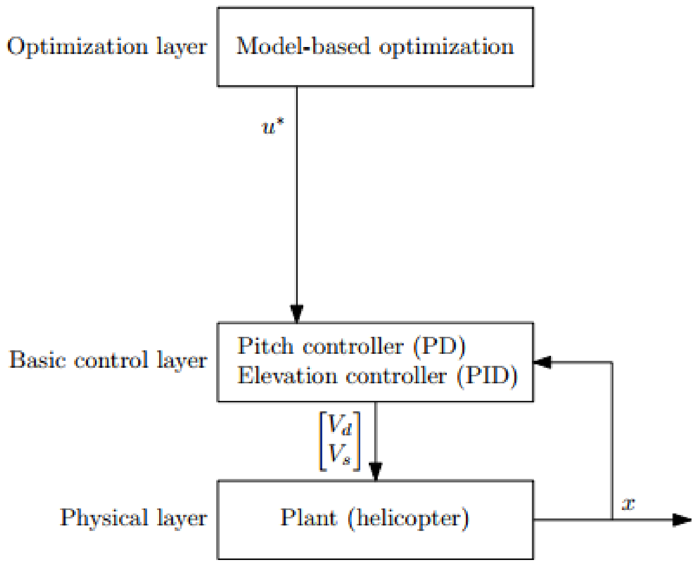


Figure 1: Layers in the control hierarchy for Task 2.

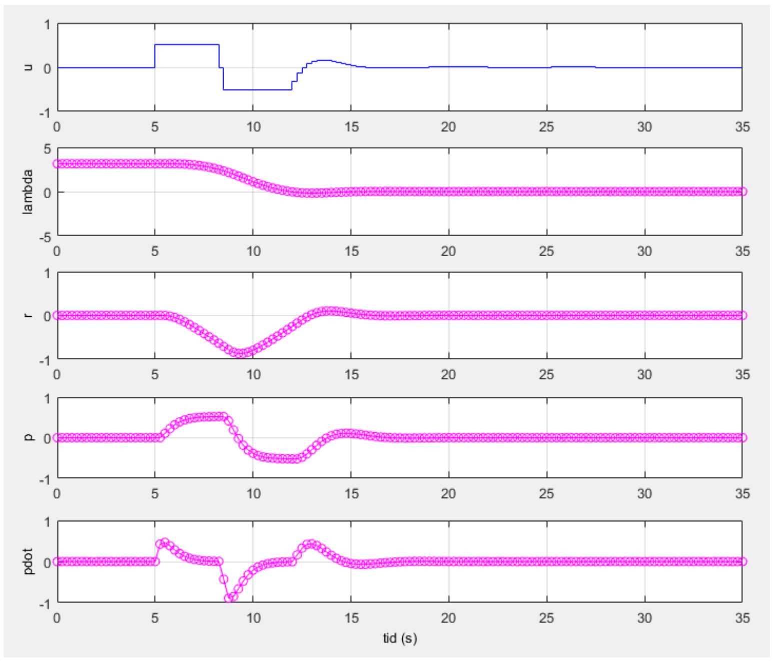


Figure 2: Optimal Trajectory with q=0.1.

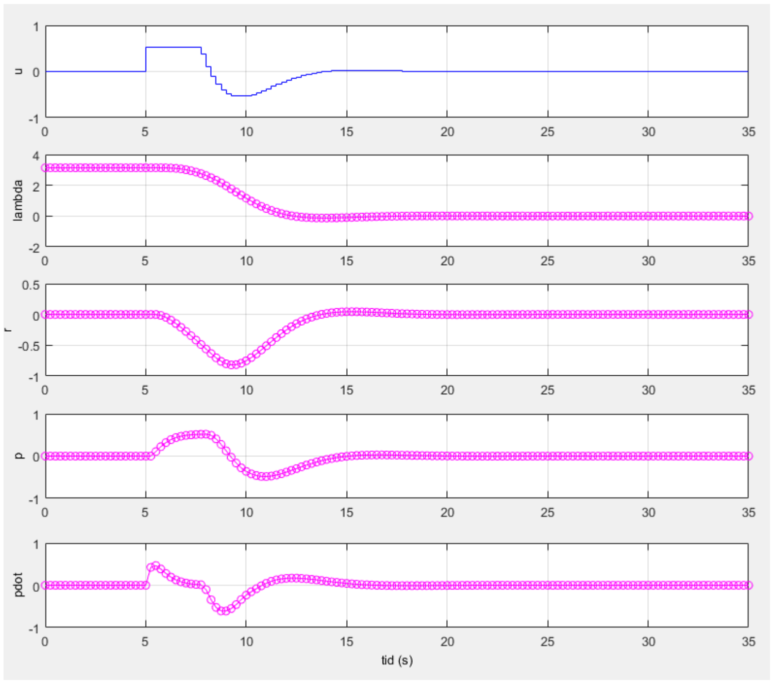


Figure 3: Optimal Trajectory with q=1.

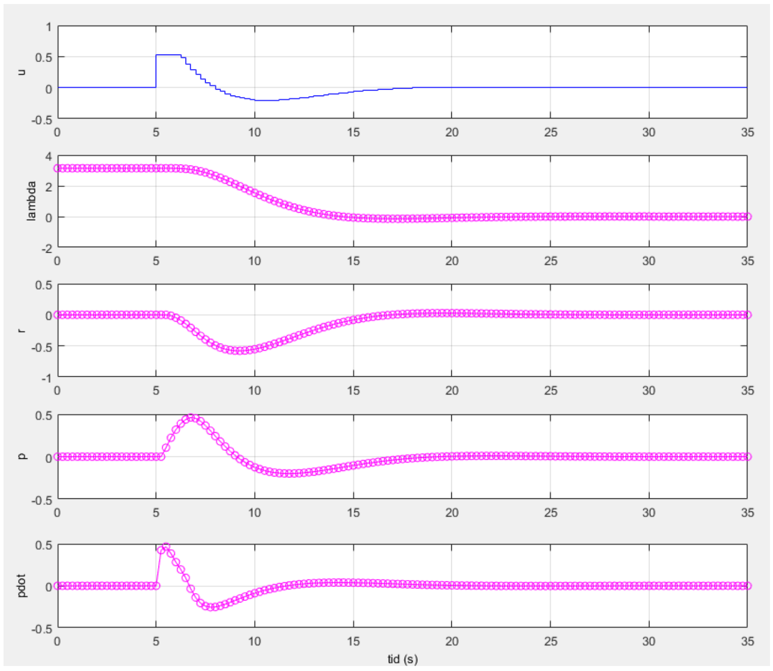


Figure 4: Optimal Trajectory with q=10.

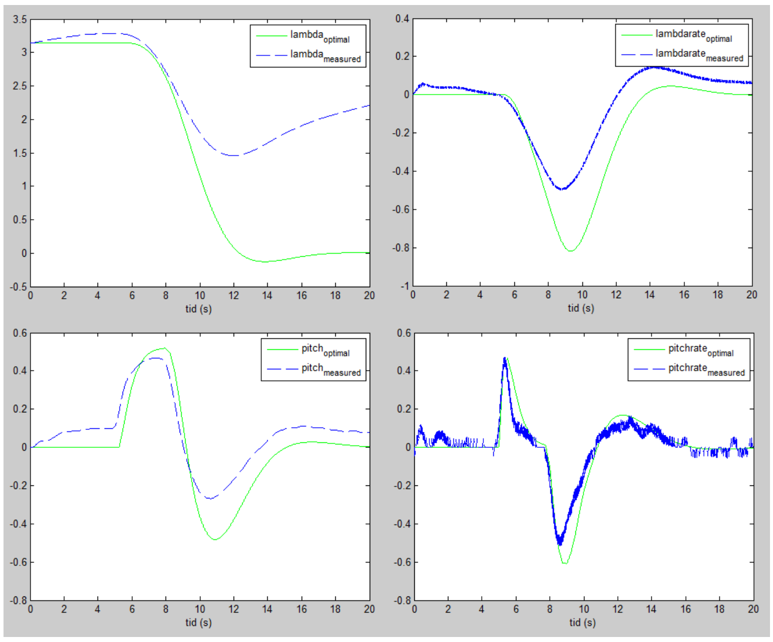


Figure 5: The calculated optimal trajectory plotted in green and the open loop response plotted in blue.

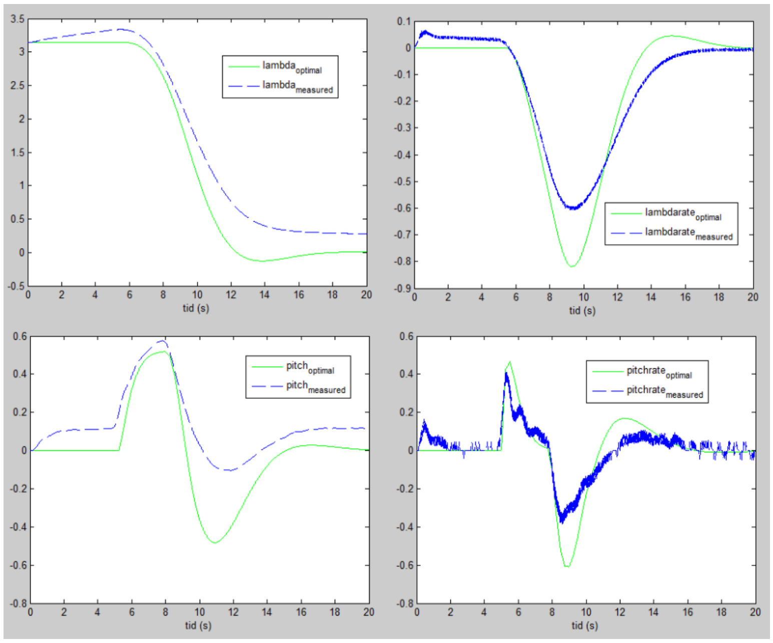


Figure 6: Closed loop response in section (4.2), before tuning cost matrices, plotted as blue. Calculated optimal trajectory plotted as green.

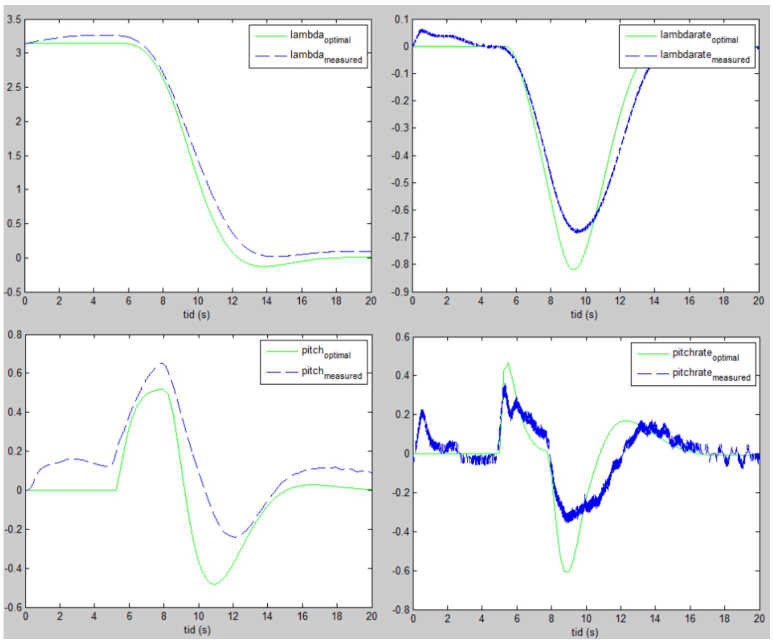


Figure 7: Closed loop response in section (4.2), after tuning cost matrices, plotted as blue. Calculated optimal trajectory plotted as green.

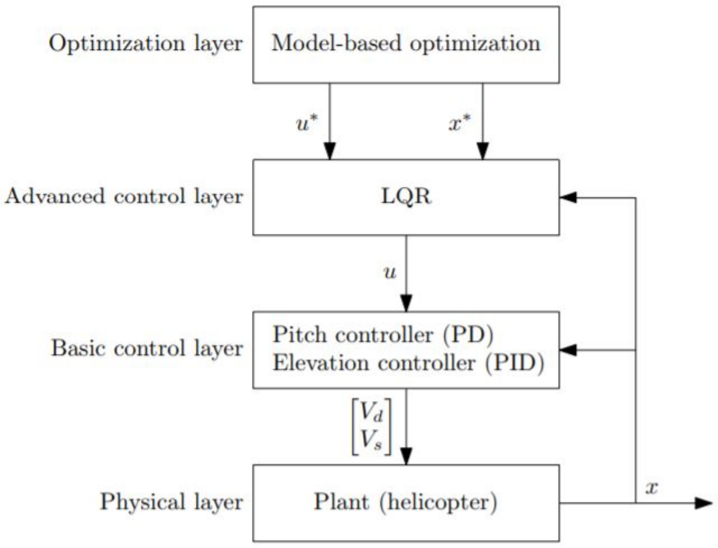


Figure 8: Control layers for the closed loop system in section (4).

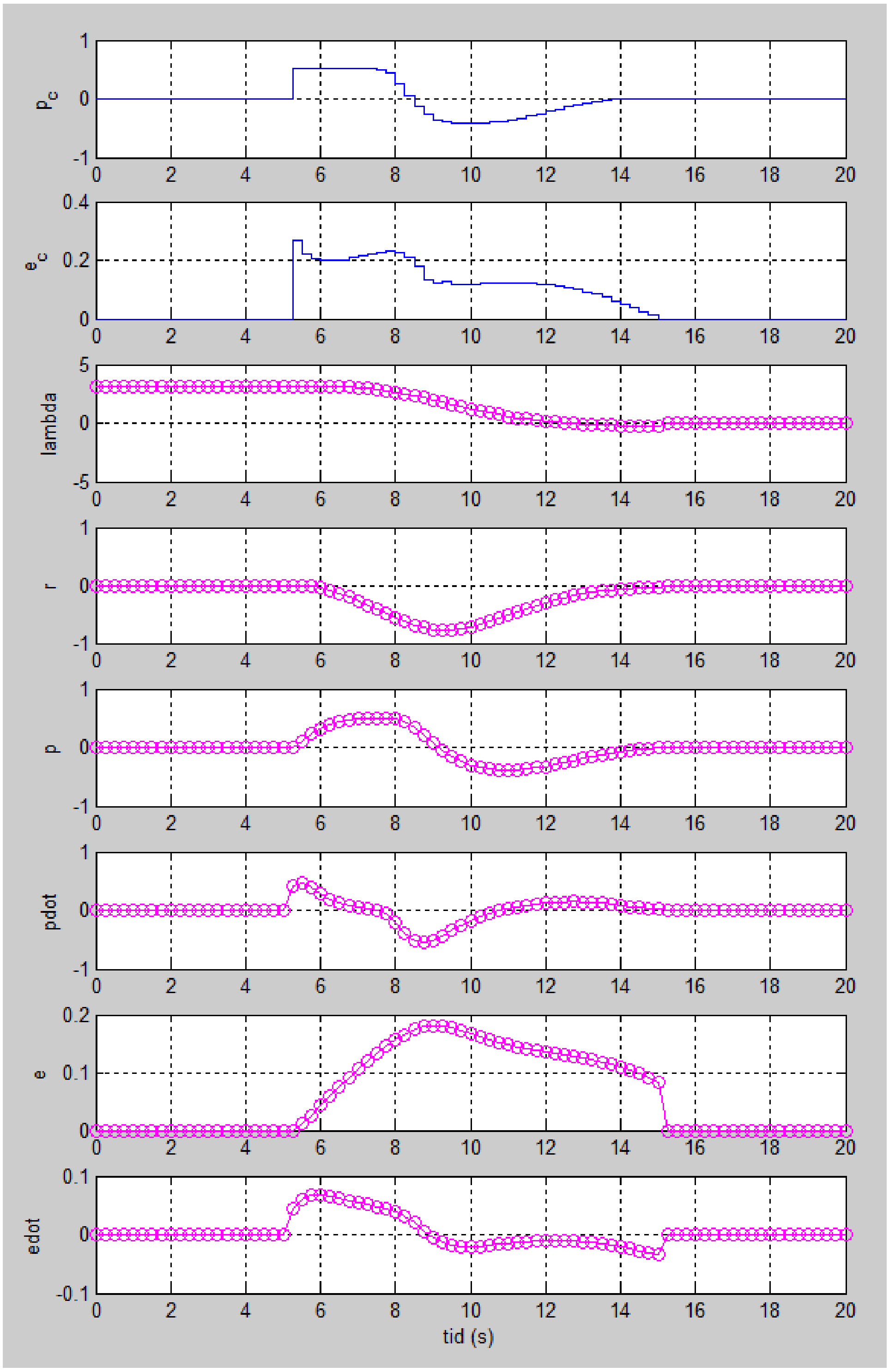


Figure 9: Calculated optimal trajectory with 6 states and nonlinear constraint on elevation.

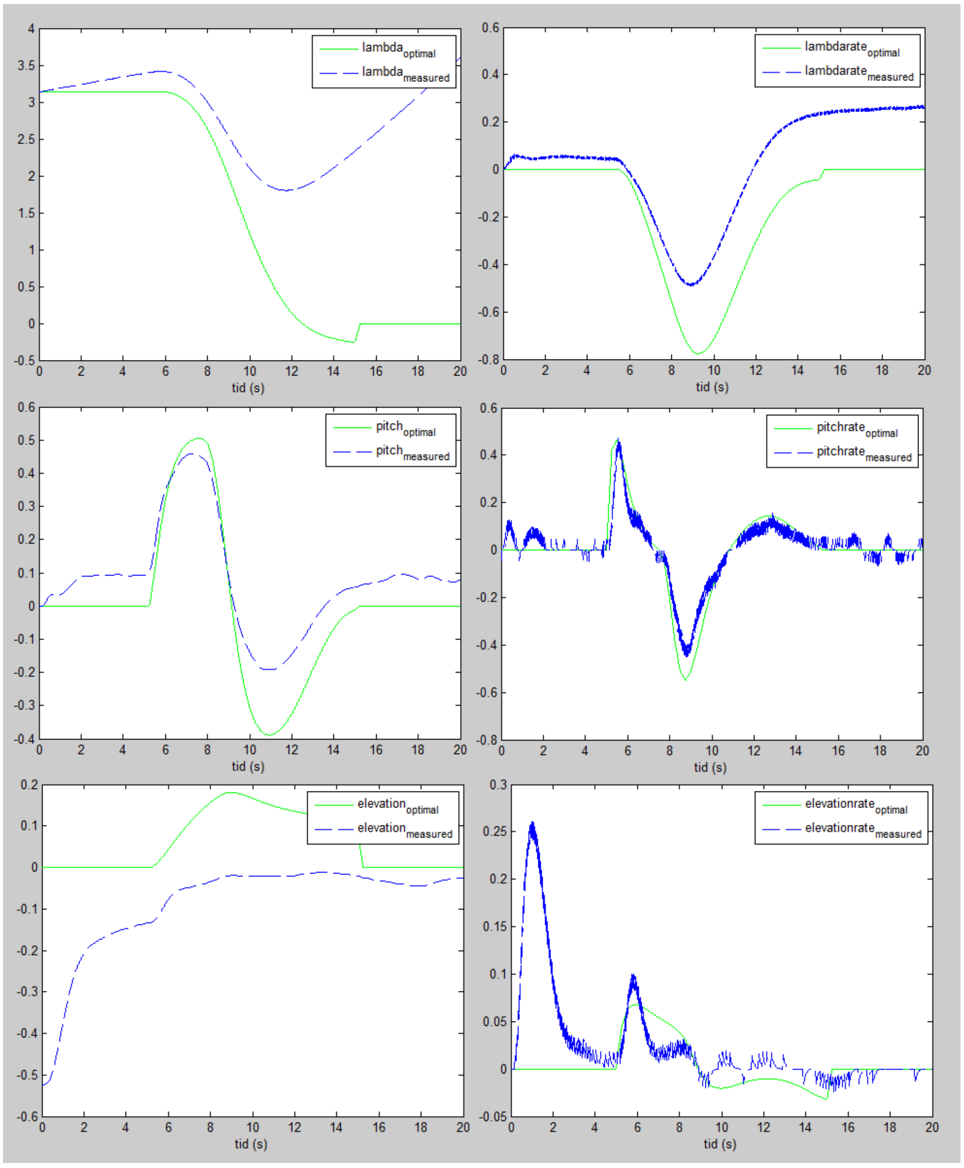


Figure 10: The calculated optimal trajectory plotted in green and the open loop response to the calculated optimal input plotted in blue.

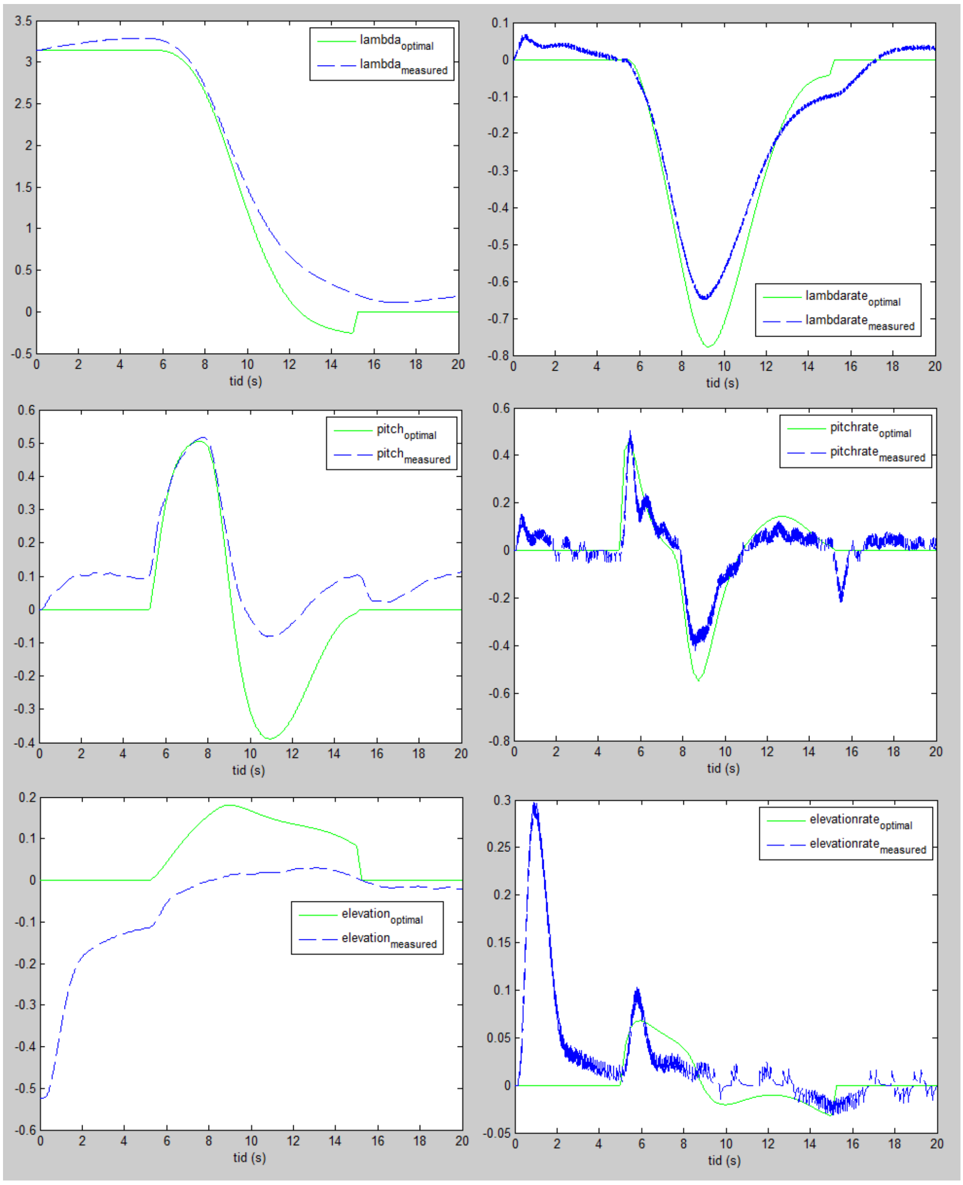


Figure 11: The calculated optimal trajectory plotted in green and the closed loop response to the calculated optimal input plotted in blue, before tuning of cost matrices.

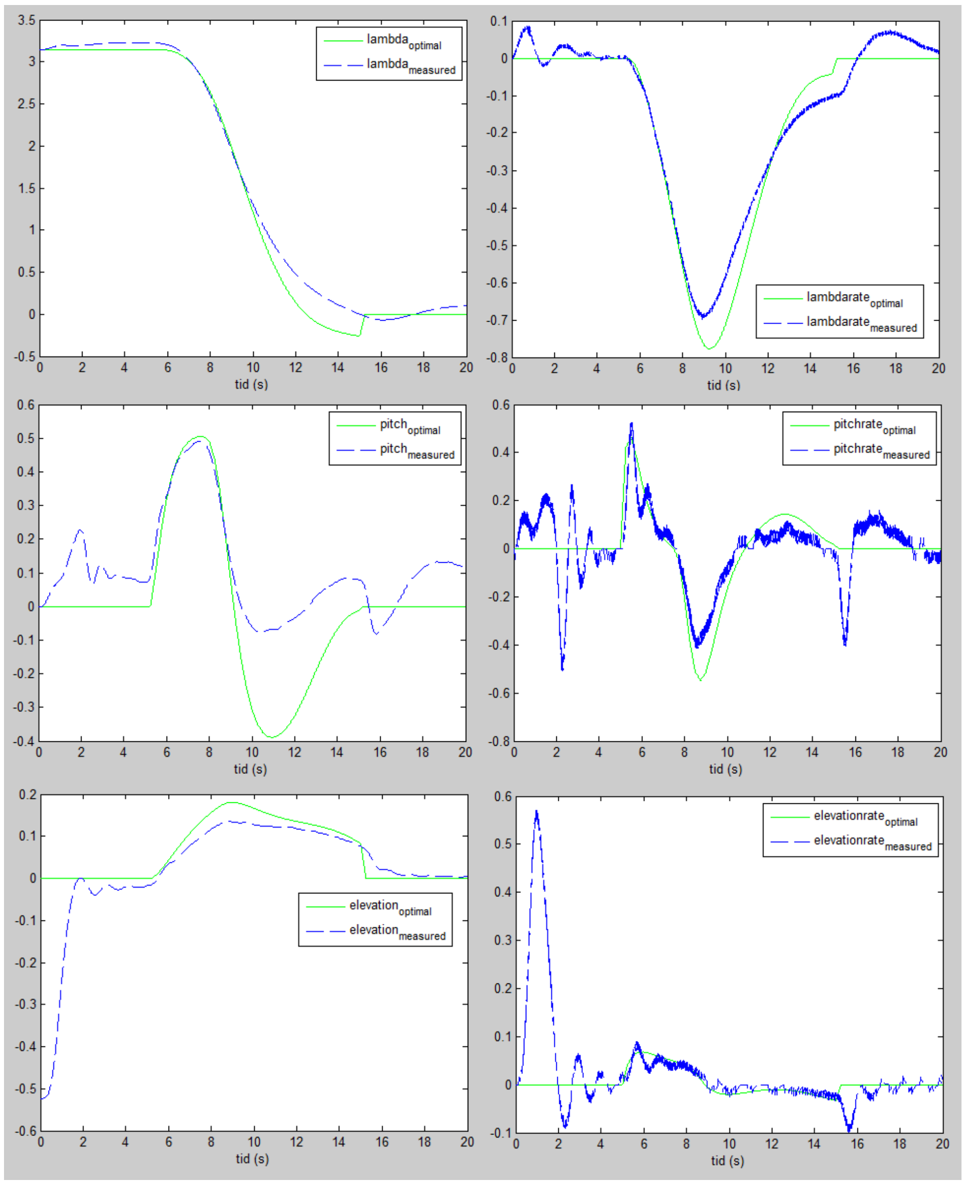


Figure 12: The calculated optimal trajectory plotted in green and the closed loop response to the calculated optimal input plotted in blue, after tuning of cost matrices.

B Simulink diagrams

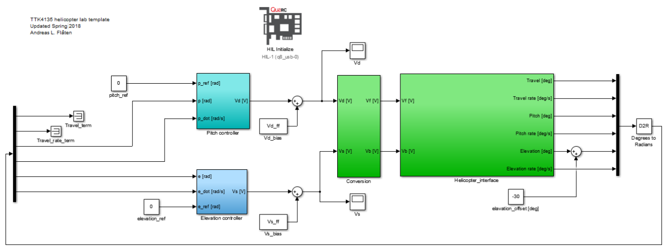


Figure 13: Initial Simulink diagram given in the assignment

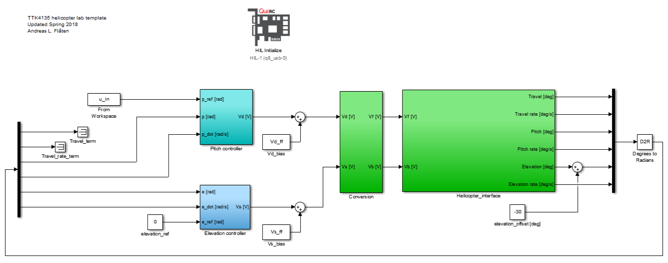


Figure 14: Simulink diagram for section (3.4).

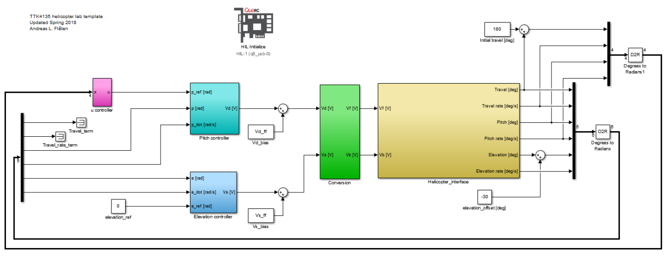


Figure 15: Simulink diagram for closed loop in section (4).

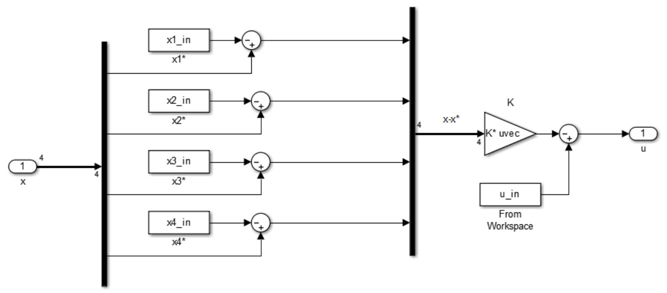


Figure 16: Simulink subsystem for closed loop in section (4).

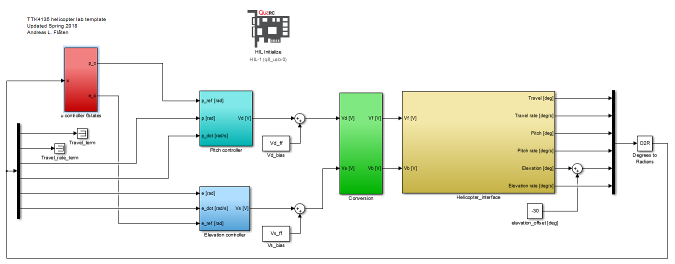


Figure 17: Simulink diagram for section (5).

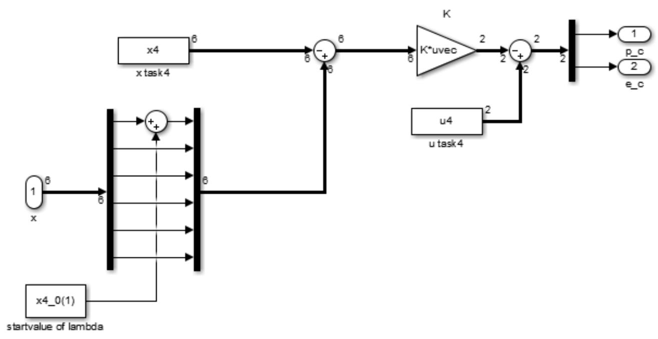
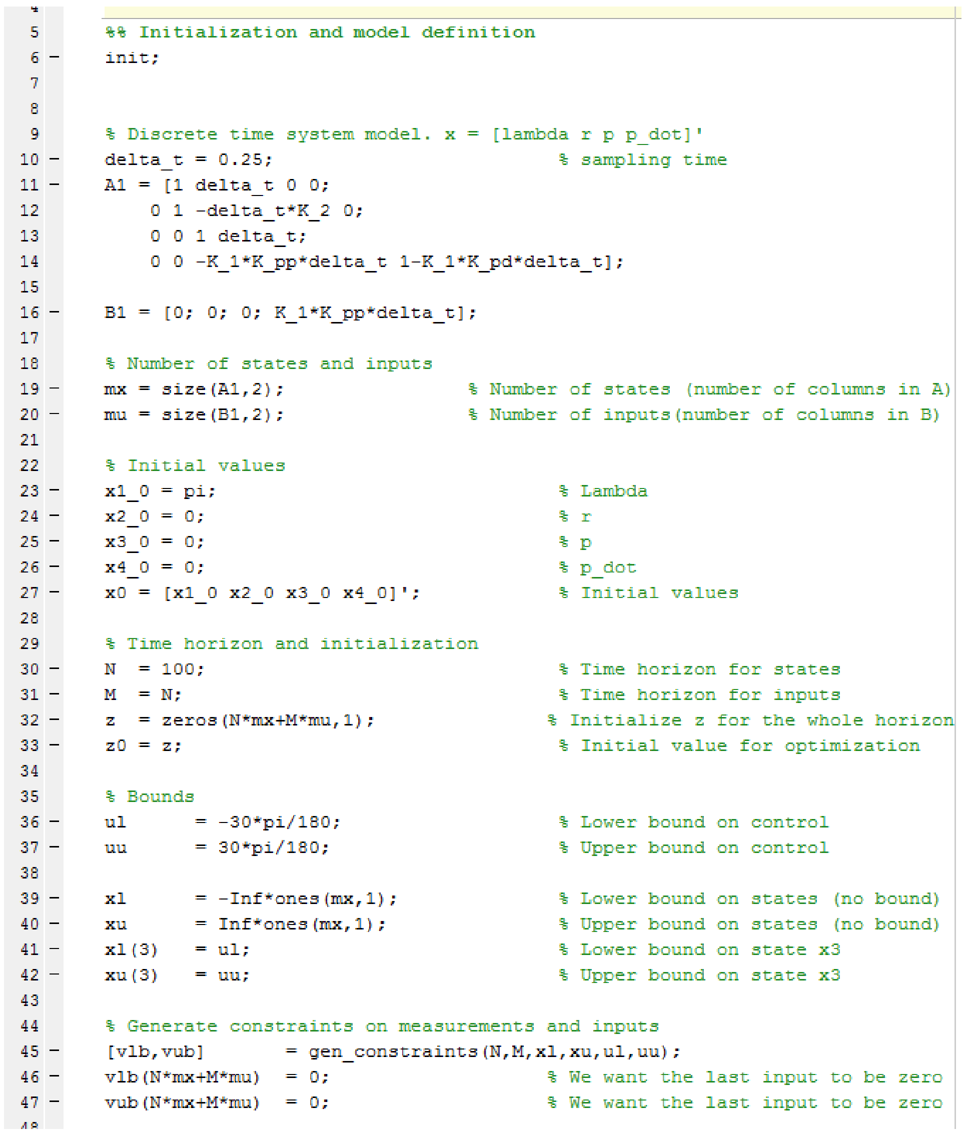


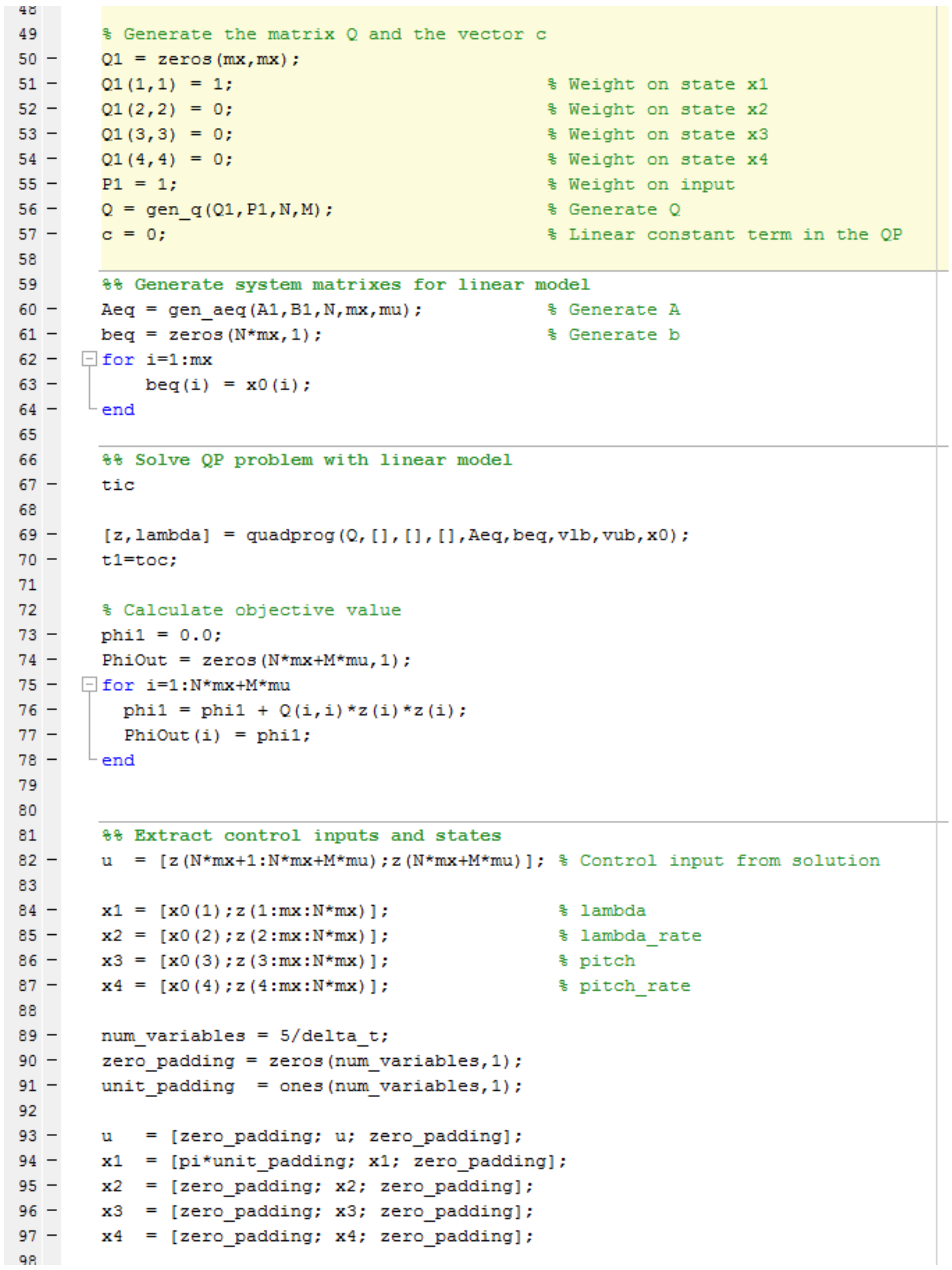
Figure 18: Simulink subsystem for section (5).

C Matlab Code



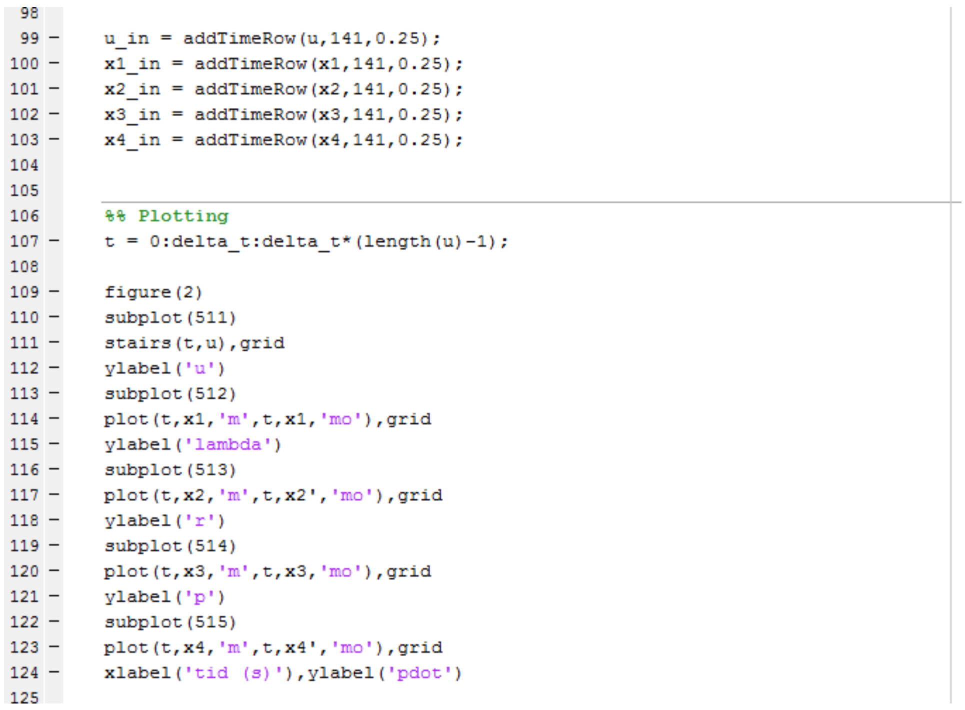
```matlab
%% Initialization and model definition
init;


% 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; K_1*K_pp*delta_t];

% 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);
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
```

Figure 19: Matlab code for Task 2 (1/3)



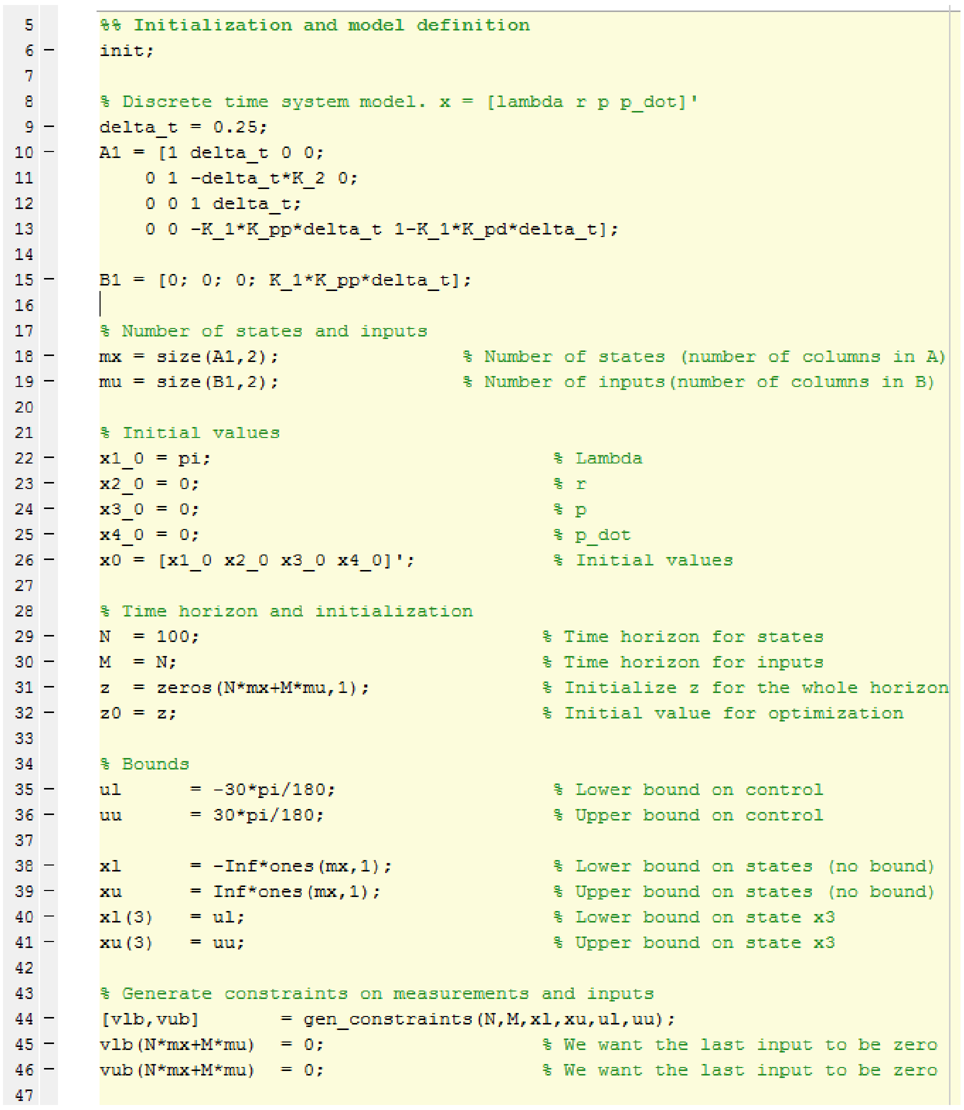
```matlab

% Generate the matrix Q and the vector c
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
c = 0;                                    % Linear constant term in the QP

%% Generate system matrixes for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu);             % Generate A
beq = zeros(N*mx,1);                      % Generate b
for i=1:mx
    beq(i) = x0(i);
end

%% Solve QP problem with linear model
tic

[z,lambda] = quadprog(Q,[],[],[],Aeq,beq,vlb,vub,x0);
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)];               % lambda
x2 = [x0(2);z(2:mx:N*mx)];               % lambda_rate
x3 = [x0(3);z(3:mx:N*mx)];               % pitch
x4 = [x0(4);z(4:mx:N*mx)];               % pitch_rate

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];

```

Figure 20: Matlab code for Task 2 (2/3)



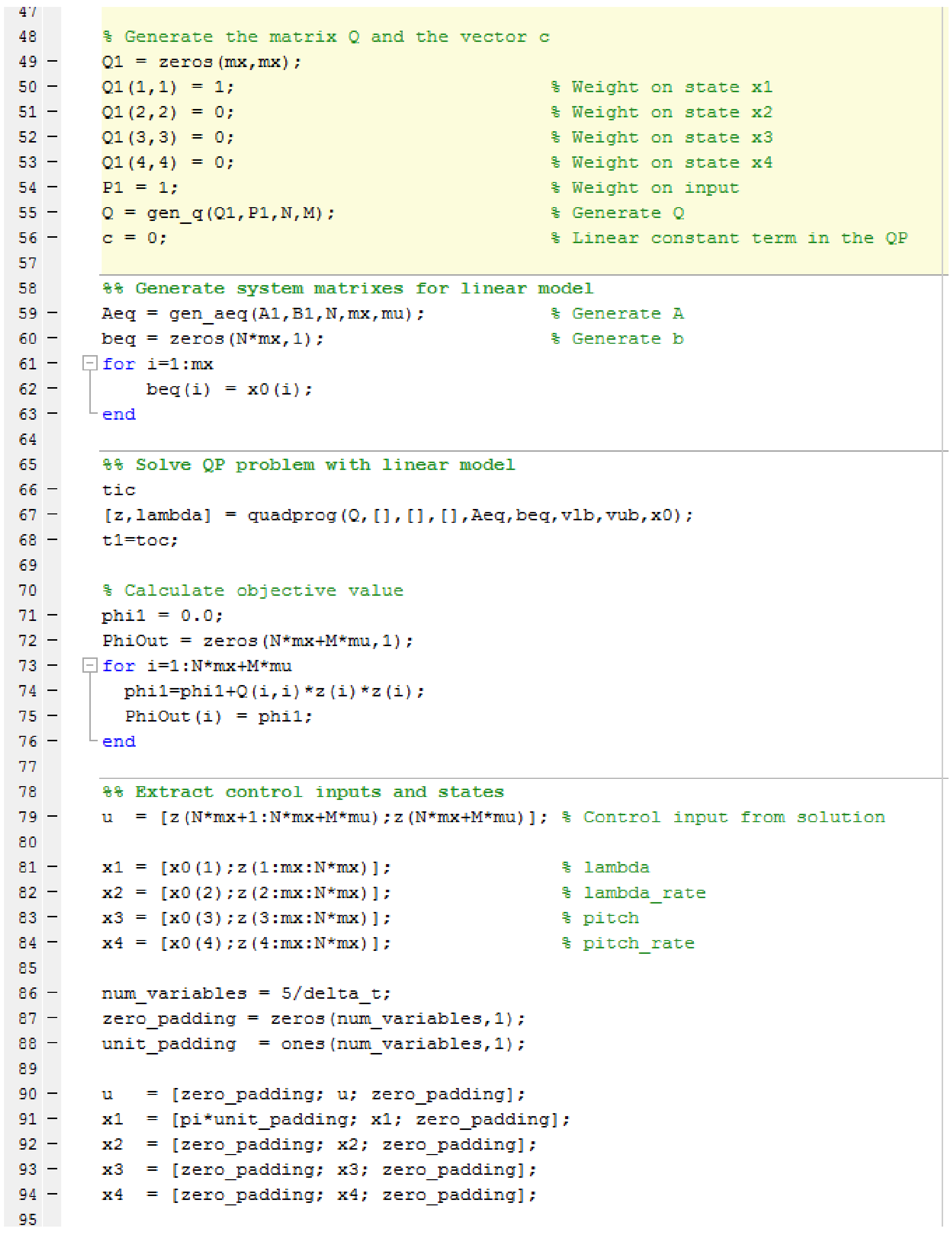
```

u_in = addTimeRow(u,141,0.25);
x1_in = addTimeRow(x1,141,0.25);
x2_in = addTimeRow(x2,141,0.25);
x3_in = addTimeRow(x3,141,0.25);
x4_in = addTimeRow(x4,141,0.25);


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

figure(2)
subplot(511)
stairs(t,u),grid
ylabel('u')
subplot(512)
plot(t,x1,'m',t,x1,'mo'),grid
ylabel('lambda')
subplot(513)
plot(t,x2,'m',t,x2','mo'),grid
ylabel('r')
subplot(514)
plot(t,x3,'m',t,x3,'mo'),grid
ylabel('p')
subplot(515)
plot(t,x4,'m',t,x4','mo'),grid
xlabel('tid (s)'),ylabel('pdot')

```

Figure 21: Matlab code for Task 2 (3/3)



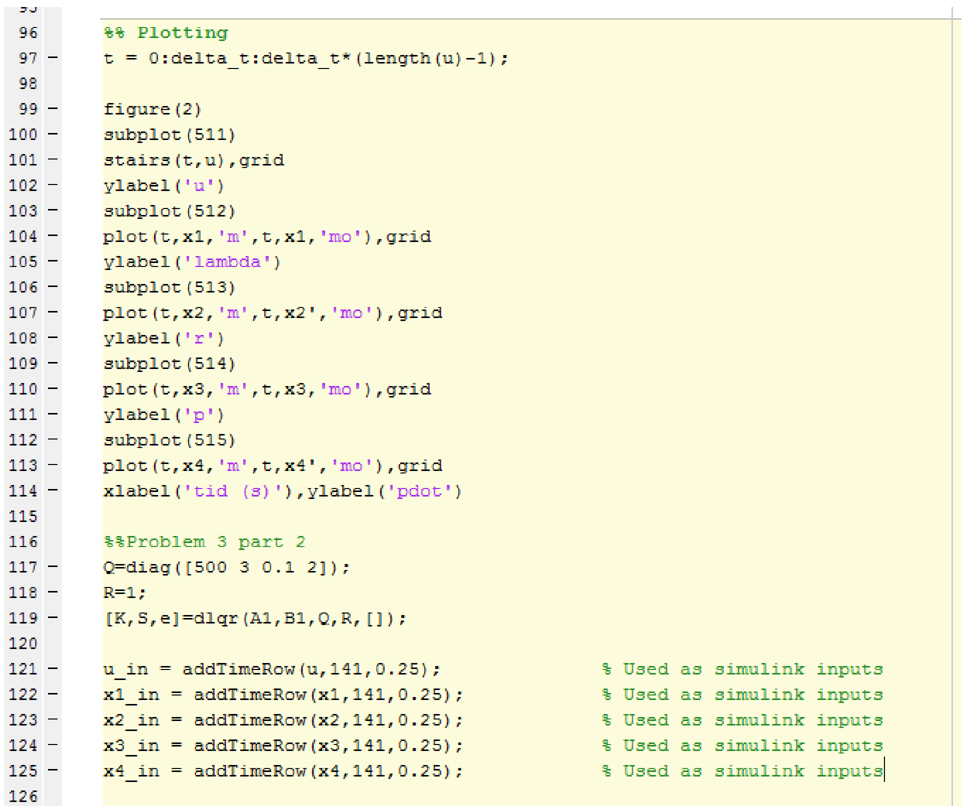
```matlab
%% Initialization and model definition
init;

% Discrete time system model. x = [lambda r p p_dot]'
delta_t = 0.25;
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; K_1*K_pp*delta_t];

% 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);
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
```

Figure 22: Matlab code for Task 3 (1/3)



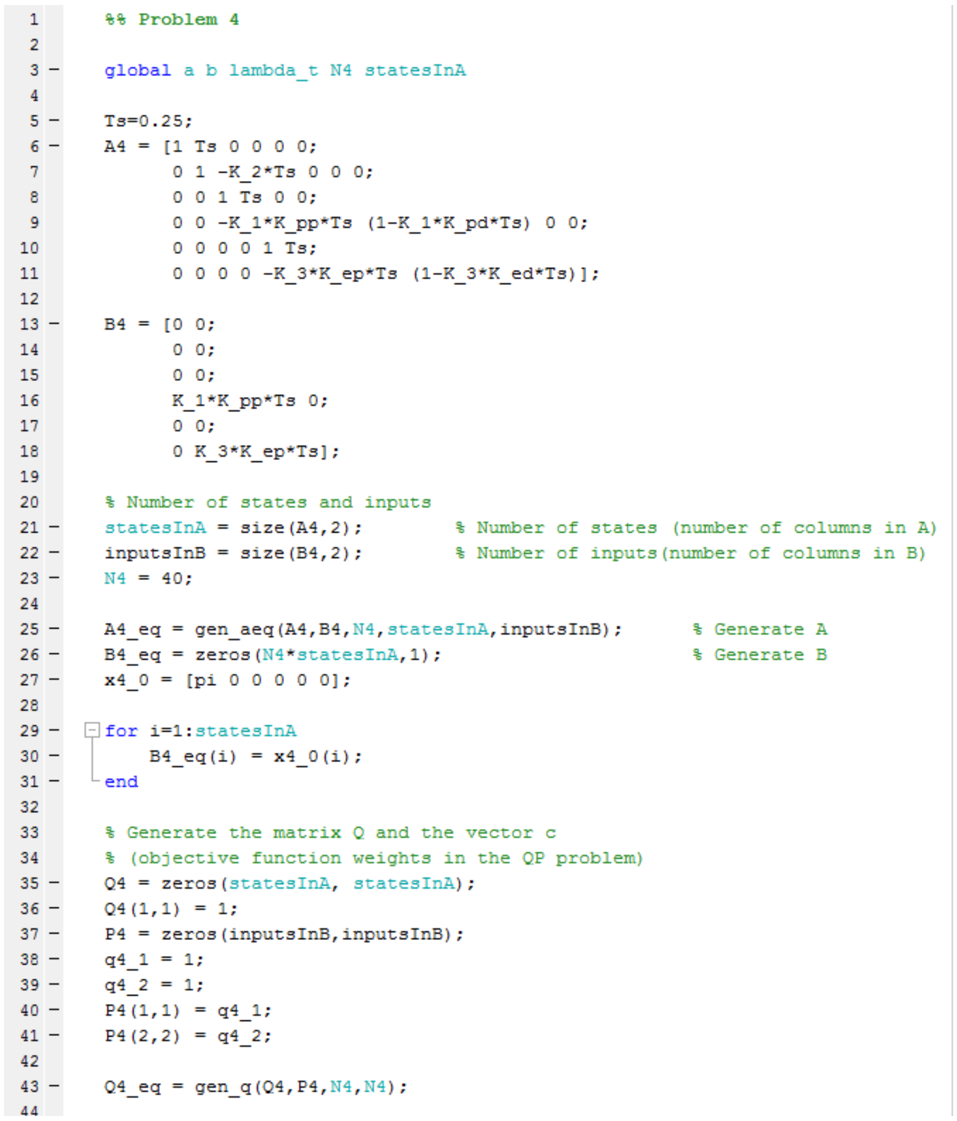
```matlab

% Generate the matrix Q and the vector c
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
c = 0;                                  % Linear constant term in the QP

%% Generate system matrixes for linear model
Aeq = gen_aeq(A1,B1,N,mx,mu);           % Generate A
beq = zeros(N*mx,1);                    % Generate b
for i=1:mx
    beq(i) = x0(i);
end

%% Solve QP problem with linear model
tic
[z,lambda] = quadprog(Q,[],[],[],Aeq,beq,vlb,vub,x0);
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)];               % lambda
x2 = [x0(2);z(2:mx:N*mx)];               % lambda_rate
x3 = [x0(3);z(3:mx:N*mx)];               % pitch
x4 = [x0(4);z(4:mx:N*mx)];               % pitch_rate

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];

```

Figure 23: Matlab code for Task 3 (2/3)



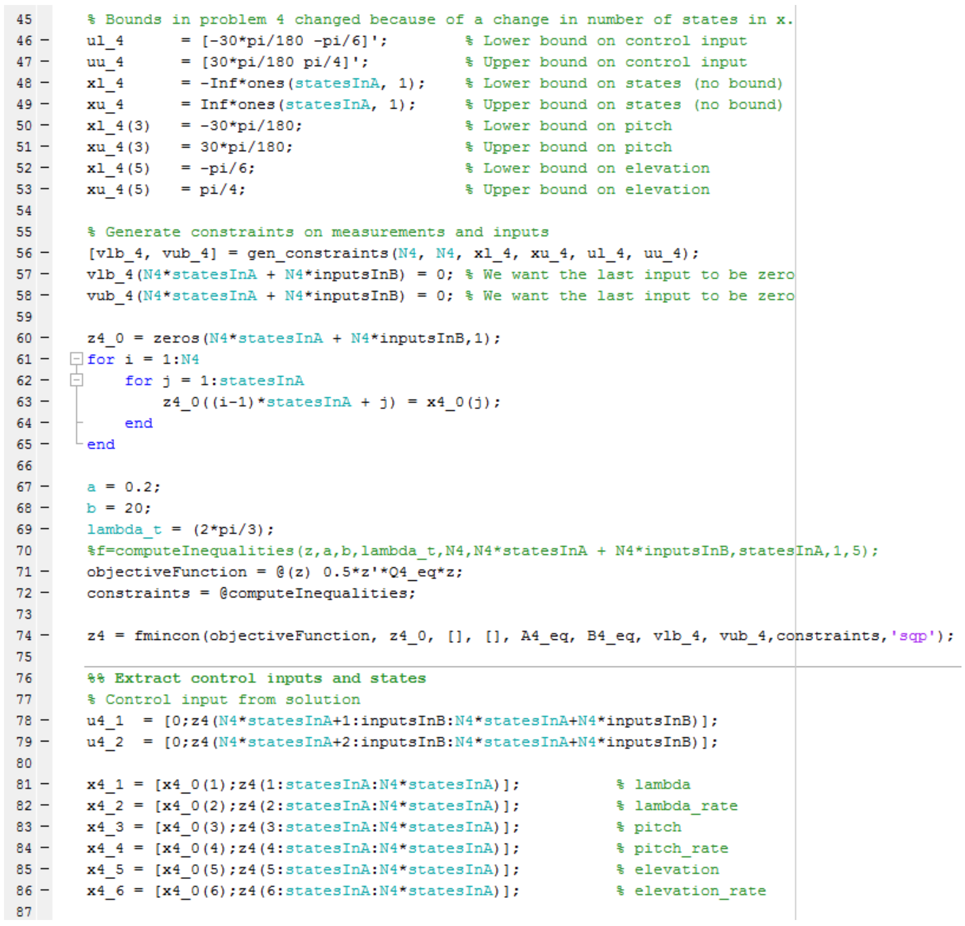
```matlab
%% Plotting
t = 0:delta_t:delta_t*(length(u)-1);

figure(2)
subplot(511)
stairs(t,u),grid
ylabel('u')
subplot(512)
plot(t,x1,'m',t,x1,'mo'),grid
ylabel('lambda')
subplot(513)
plot(t,x2,'m',t,x2','mo'),grid
ylabel('r')
subplot(514)
plot(t,x3,'m',t,x3,'mo'),grid
ylabel('p')
subplot(515)
plot(t,x4,'m',t,x4','mo'),grid
xlabel('tid (s)'),ylabel('pdot')

%%Problem 3 part 2
Q=diag([500 3 0.1 2]);
R=1;
[K,S,e]=dlqr(A1,B1,Q,R,[]);

u_in = addTimeRow(u,141,0.25);            % Used as simulink inputs
x1_in = addTimeRow(x1,141,0.25);          % Used as simulink inputs
x2_in = addTimeRow(x2,141,0.25);          % Used as simulink inputs
x3_in = addTimeRow(x3,141,0.25);          % Used as simulink inputs
x4_in = addTimeRow(x4,141,0.25);          % Used as simulink inputs
```

Figure 24: Matlab code for Task 3 (3/3)



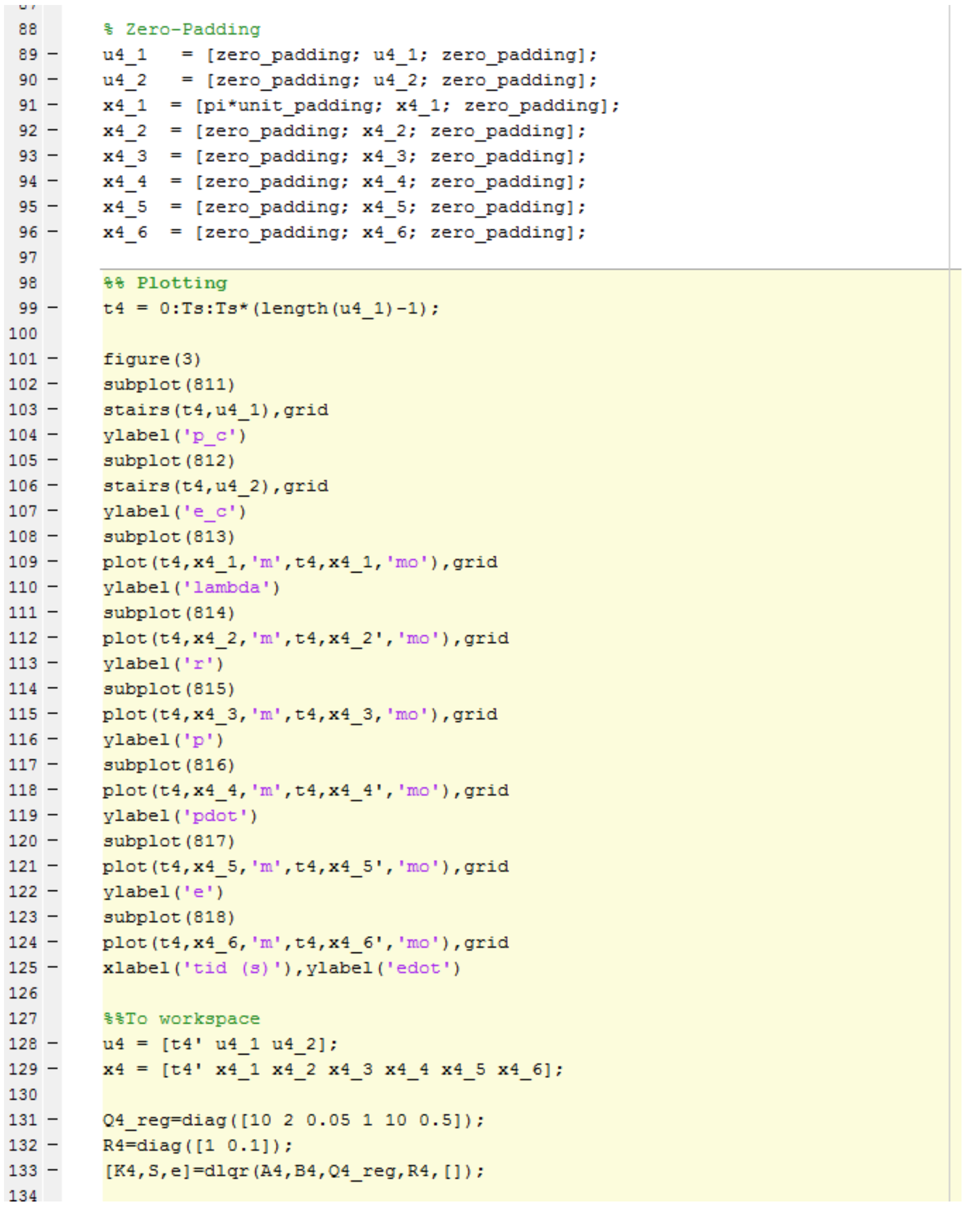
```matlab
%% Problem 4

global a b lambda_t N4 statesInA

Ts=0.25;
A4 = [1 Ts 0 0 0 0;
      0 1 -K_2*Ts 0 0 0;
      0 0 1 Ts 0 0;
      0 0 -K_1*K_pp*Ts (1-K_1*K_pd*Ts) 0 0;
      0 0 0 0 1 Ts;
      0 0 0 0 -K_3*K_ep*Ts (1-K_3*K_ed*Ts)];

B4 = [0 0;
      0 0;
      0 0;
      K_1*K_pp*Ts 0;
      0 0;
      0 K_3*K_ep*Ts];

% Number of states and inputs
statesInA = size(A4,2);        % Number of states (number of columns in A)
inputsInB = size(B4,2);        % Number of inputs(number of columns in B)
N4 = 40;

A4_eq = gen_aeq(A4,B4,N4,statesInA,inputsInB);      % Generate A
B4_eq = zeros(N4*statesInA,1);                      % Generate B
x4_0 = [pi 0 0 0 0 0];

for i=1:statesInA
    B4_eq(i) = x4_0(i);
end

% Generate the matrix Q and the vector c
% (objective function weights in the QP problem)
Q4 = zeros(statesInA, statesInA);
Q4(1,1) = 1;
P4 = zeros(inputsInB,inputsInB);
q4_1 = 1;
q4_2 = 1;
P4(1,1) = q4_1;
P4(2,2) = q4_2;

Q4_eq = gen_q(Q4,P4,N4,N4);
```

Figure 25: Matlab code for Task 4 (1/3)



```matlab
% Bounds in problem 4 changed because of a change in number of states in x.
ul_4      = [-30*pi/180 -pi/6]';          % Lower bound on control input
uu_4      = [30*pi/180 pi/4]';            % Upper bound on control input
xl_4      = -Inf*ones(statesInA, 1);      % Lower bound on states (no bound)
xu_4      = Inf*ones(statesInA, 1);       % Upper bound on states (no bound)
xl_4(3)   = -30*pi/180;                   % Lower bound on pitch
xu_4(3)   = 30*pi/180;                    % Upper bound on pitch
xl_4(5)   = -pi/6;                        % Lower bound on elevation
xu_4(5)   = pi/4;                         % Upper bound on elevation

% Generate constraints on measurements and inputs
[vlb_4, vub_4] = gen_constraints(N4, N4, xl_4, xu_4, ul_4, uu_4);
vlb_4(N4*statesInA + N4*inputsInB) = 0; % We want the last input to be zero
vub_4(N4*statesInA + N4*inputsInB) = 0; % We want the last input to be zero

z4_0 = zeros(N4*statesInA + N4*inputsInB,1);
for i = 1:N4
    for j = 1:statesInA
        z4_0((i-1)*statesInA + j) = x4_0(j);
    end
end

a = 0.2;
b = 20;
lambda_t = (2*pi/3);
%f=computeInequalities(z,a,b,lambda_t,N4,N4*statesInA + N4*inputsInB,statesInA,1,5);
objectiveFunction = @(z) 0.5*z'*Q4_eq*z;
constraints = @computeInequalities;

z4 = fmincon(objectiveFunction, z4_0, [], [], A4_eq, B4_eq, vlb_4, vub_4,constraints,'sqp');

%% Extract control inputs and states
% Control input from solution
u4_1  = [0;z4(N4*statesInA+1:inputsInB:N4*statesInA+N4*inputsInB)];
u4_2  = [0;z4(N4*statesInA+2:inputsInB:N4*statesInA+N4*inputsInB)];

x4_1 = [x4_0(1);z4(1:statesInA:N4*statesInA)];          % lambda
x4_2 = [x4_0(2);z4(2:statesInA:N4*statesInA)];          % lambda_rate
x4_3 = [x4_0(3);z4(3:statesInA:N4*statesInA)];          % pitch
x4_4 = [x4_0(4);z4(4:statesInA:N4*statesInA)];          % pitch_rate
x4_5 = [x4_0(5);z4(5:statesInA:N4*statesInA)];          % elevation
x4_6 = [x4_0(6);z4(6:statesInA:N4*statesInA)];          % elevation_rate
```

Figure 26: Matlab code for Task 4 (2/3)



```
% Zero-Padding
u4_1   = [zero_padding; u4_1; zero_padding];
u4_2   = [zero_padding; u4_2; zero_padding];
x4_1  = [pi*unit_padding; x4_1; zero_padding];
x4_2  = [zero_padding; x4_2; zero_padding];
x4_3  = [zero_padding; x4_3; zero_padding];
x4_4  = [zero_padding; x4_4; zero_padding];
x4_5  = [zero_padding; x4_5; zero_padding];
x4_6  = [zero_padding; x4_6; zero_padding];

%% Plotting
t4 = 0:Ts:Ts*(length(u4_1)-1);

figure(3)
subplot(811)
stairs(t4,u4_1),grid
ylabel('p_c')
subplot(812)
stairs(t4,u4_2),grid
ylabel('e_c')
subplot(813)
plot(t4,x4_1,'m',t4,x4_1,'mo'),grid
ylabel('lambda')
subplot(814)
plot(t4,x4_2,'m',t4,x4_2','mo'),grid
ylabel('r')
subplot(815)
plot(t4,x4_3,'m',t4,x4_3,'mo'),grid
ylabel('p')
subplot(816)
plot(t4,x4_4,'m',t4,x4_4','mo'),grid
ylabel('pdot')
subplot(817)
plot(t4,x4_5,'m',t4,x4_5','mo'),grid
ylabel('e')
subplot(818)
plot(t4,x4_6,'m',t4,x4_6','mo'),grid
xlabel('tid (s)'),ylabel('edot')

%%To workspace
u4 = [t4' u4_1 u4_2];
x4 = [t4' x4_1 x4_2 x4_3 x4_4 x4_5 x4_6];

Q4_reg=diag([10 2 0.05 1 10 0.5]);
R4=diag([1 0.1]);
[K4,S,e]=dlqr(A4,B4,Q4_reg,R4,[]);
```

Figure 27: Matlab code for Task 4 (3/3)