Ball Beam 3

Mechatronics - State Space Control

Jack Gibson, Andreas Wiedholz, Jonathan Ott

Contents

Introduction

The aim of this project is to control a non-linear electro-mechanical system, widely referred to as the "ball and beam". The ball is placed at some point along the beam and the system shall be controlled to stabilise it. The basic model, shown in Figure 1 shows a simplified diagram of the system. The system shall be controlled using various control techniques, including state feedback and PID control.

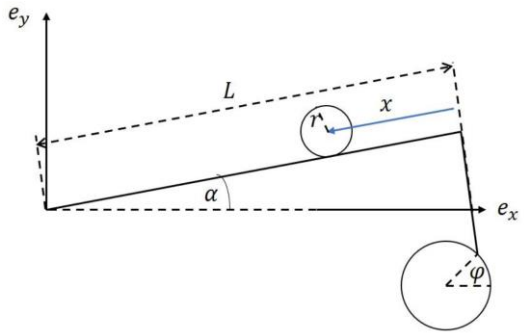

Figure 1 Simplified model of the ball beam.

In Figure 1, it can be observed that x and α describe the model parameters for displacement of the ball and rotation of the beam, respectively. In our state space model, we require four states and so the first derivative of both these parameters are taken to give a ball velocity and an angular velocity. The vector below shows these states:

$$\begin{bmatrix} x \\ \dot{x} \\ \alpha \\ \dot{\alpha} \end{bmatrix}$$

The Nonlinear model has been converted into a state space representation by linearising the equations shown under the Nonlinear header.

$$\dot{x} = Ax + Bu = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0.0290 & 0 & 7.0062 & 0 \\ 0 & 0 & 0 & 1 \\ -3.1991 & 0 & 0.1088 & 0 \end{bmatrix} x + \begin{bmatrix} 0 \\ -4.332 \\ 0 \\ 47.7514 \end{bmatrix} u$$

$$y = Cx + Du = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} x$$

Nonlinear System

The first task is to represent the Nonlinear model in Simulink. As we use 2 outputs (x and α) we require 2 equations to model this.

$$\ddot{x} = -\frac{m(l-x)(\dot{\alpha})^2 - gm\,.\sin\alpha + \frac{mr(\tau + 2m(l-x)\dot{x}\alpha + gm(\mathrm{x}\cos\alpha - \dot{l}\cos\alpha + r\sin\alpha))}{J + m((l-x)^2 + r^2)}}{\frac{2m}{5} - m(\frac{2mr^2}{J + m((l-x)^2 + r^2)} - 1)}$$

$$\ddot{\alpha} = \frac{\tau - m(r(\frac{5g\sin\alpha}{7} + \frac{5x(\dot{\alpha})^2}{7} - \frac{5l(\dot{\alpha})^2}{7}) - 2(l-x)\dot{x}\dot{\alpha}) + gm(x\cos\alpha - l\cos\alpha + r\sin\alpha)}{J + m((l-x)^2 + \frac{2r^2}{7})}$$

The Nonlinear system model can be found in Figure 22 and Figure 23 later in the report.

Jack Gibson, Andreas Wiedholz, Jonathan Ott

After implementing these equations in a Simulink model, we set initial conditions to the integrators, α= 0.1rad and x =L/2 to drop the ball at the midpoint of the beam. To ensure a more critical assessment of our system, we set some boundaries to our model to stop the simulation if the ball should fall off the beam (i.e., x<0 or x>L). The behaviour we observe in Figure 2 shows one we would expect. The uncontrolled system has no ability to correct itself and so the ball falls off and the beam comes to rest at an angle of ~90°.

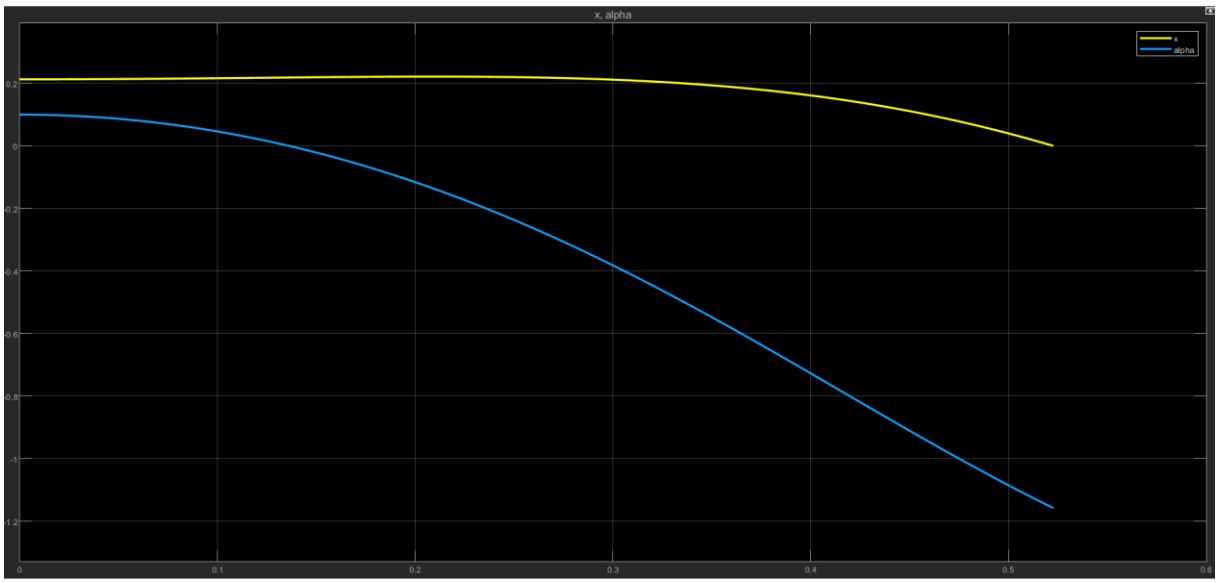

Figure 2 Behaviour of uncontrolled Nonlinear system

System Analysis

It is vital that when modelling a system, the engineer understands the poles and zeros present, as well as the controllability and observability of the system. For a continuous time model, we can assume that any pole with a positive real part can be regarded as terminal and will ensure the system goes unstable. The system initialised below shows the Linearised state space model.

```
A = [0 1 0 0; 0.0290 0 7.0062 0; 0 0 0 1; -3.1991 0 0.1088 0];
B = [0; -4.332; 0; 47.7514];
C = [1 0 0 0; 0 0 1 0];
D = [0];
sys = ss(A, B, C, D)
```

The poles and zeros of the system can be mapped in MATLAB by using the pzmap function.

```
[p, z] = pzmap(sys)
```

It is conclusive that there are 4 poles in the system lying at:

1. $-1.5498 + 1.5274i$
2. $-1.5498 - 1.5274i$
3. $1.5498 + 1.5274i$
4. $1.5498 - 1.5274i$

From this analysis we can see two poles with positive real parts, shown here as pole number 3 and 4. It is critical that if the system is to be stable, these 2 poles must be brought back into the negative plane.

Before designing the controller, it is also critical to determine if the system is controllable and/or observable. In order to obtain these results two matrices, Pc and Po must be created,

```
Pc = [B A*B A*A*B A*A*A*B];
```

```
Pc =

         0   -4.3320         0  334.4302
   -4.3320         0  334.4302         0
         0   47.7514         0   19.0539
   47.7514         0   19.0539         0
```

```
Po = [C; C*A; C*A*A; C*A*A*A];
```

```
Po =

    1.0000         0         0         0
         0         0    1.0000         0
         0    1.0000         0         0
         0         0         0    1.0000
    0.0290         0    7.0062         0
   -3.1991         0    0.1088         0
         0    0.0290         0    7.0062
         0   -3.1991         0    0.1088
```

We can conclude therefore that this system is both fully observable and controllable, as the rank of these matrices can be determined as 4. If any other rank is detected we must conclude that the system is uncontrollable and/or unobservable.

PID Controller

In this task a PID approach must be taken to control the system. The state space model can be converted into two transfer functions to use the PID controller. This is done simply in MATLAB with the command `tf(sys)`.

As the ball beam system has two outputs to be controlled the result of this function is two transfer functions, one for displacement (x) and another for angular rotation (α) of the beam. These transfer functions are shown below:

$$tf1 = \frac{-4.332s^2 + 1.539e-14\,s + 335}{s^4 - 0.1378s^2 + 8.882e-15\,s + 22.42}$$

$$tf2 = \frac{47.75s^2 + 12.47}{s^4 - 0.1378s^2 + 8.882e-15\,s + 22.42}$$

The most trivial and highly effective method for tuning the PID controller is to use MATLAB's SISO tool. The transfer function is first loaded into the architecture of the closed loop feedback system and the values of the PID can be manipulated to shift the poles to the desired location to achieve an acceptable response.

$$\frac{-2266.7(s+1372)^2}{s}$$

The step response of this PID in the closed loop system looks like:

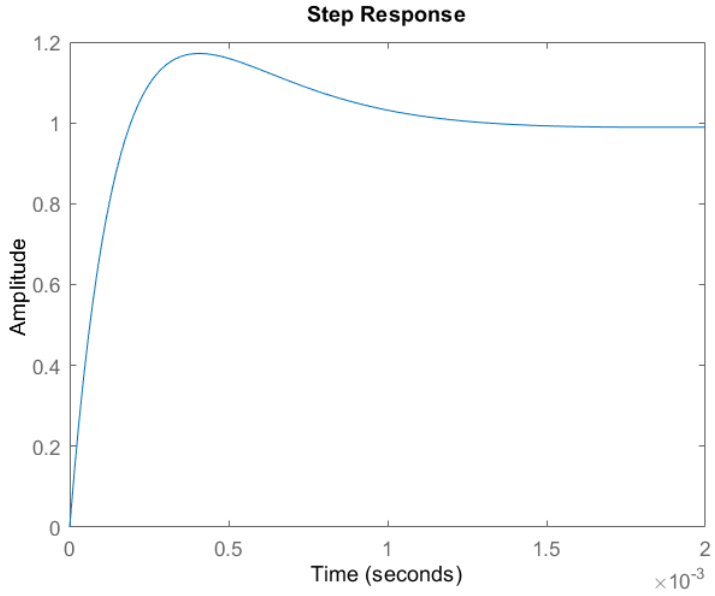

Figure 3 Stable step response for tf1

The second transfer function, used to control the angular displacement of the beam, can then also be loaded into the SISO tool environment to produce a PID controller shown in figure 4 below.

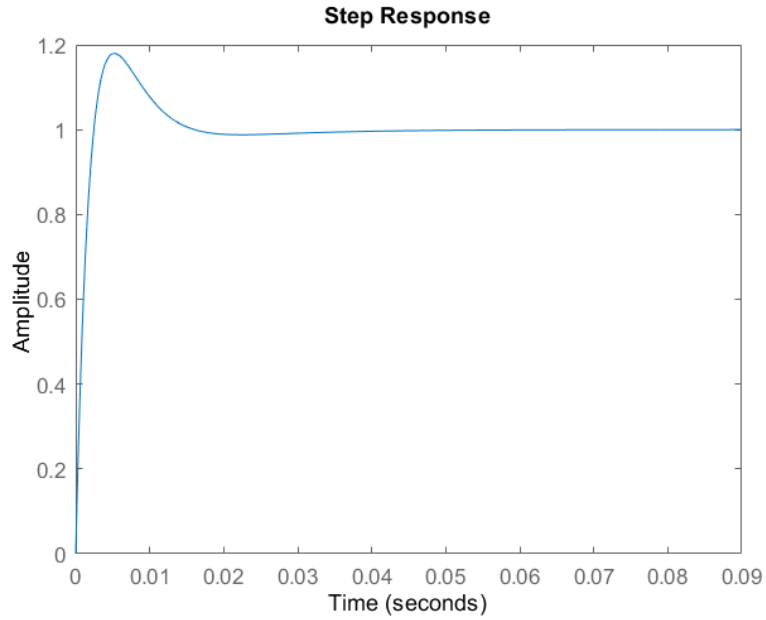

Figure 4 Stable step response for tf2

The step response of this PID in the closed loop system looks like:

$$\frac{15.773\ (s + 111.8)^2}{s}$$

The results of both controllers are to be accepted as the step response settles to a final value with no steady state error, with very little overshoot and a fast settling time.

Full State Feedback

The aim of designing a controller based on the Linear state space system, is to obtain a working controller, where the dynamics of the system can be understood better. Once an effective controller is designed for the linear system, it can then be implemented on the Non-Linear model which resembles the actual system more accurately.

In order to design a Full State Feedback controller (FSFB) several methods can be used, from which Pole Placement and Linear Quadratic Regulator (LQR) were chosen. In both methods the main aim remains, to find a good value for the feedback gain to stabilise the system. To remain consistent with the Nonlinear model, upper and lower limits have been defined, to stop the simulation if the ball rolls outside of the limits of the beam.

In the state space model shown in Figure 5 the 'A' and 'B' gain blocks represent the state and the input matrix respectively. The C gain refers to the output matrix of a state space model. Gain K is the Full State Feedback vector containing the tuned gains for each state. The initial conditions of the integrator 'x' must be changed in order to better represent the system. Values of [L/2; 0; 0.1; 0] are chosen for each state. A displacement of L/2 is chosen to make the ball drop onto the centre point of the beam. Both velocities are set to zero to replicate the start of the system at rest and the angular displacement is set to 0.1 rads to show the slight rotation of angle α of the beam.

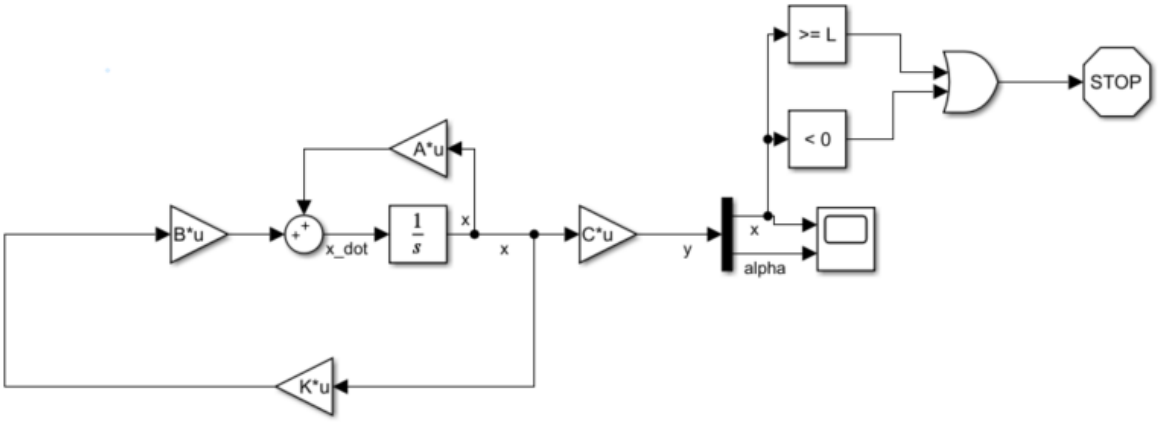

Figure 5 Simulink model for full state feedback

In the script BallBeam3.m there is a subsection called *Full State Feedback.* Within this section there is a variable called use_pp. By setting this variable to either 'true' or 'false' the user can switch between pole placement or the LQR design method to calculate the control gain K and for later use on the compensator to calculate the observer gain L.

Pole Placement

Pole Placement is a method of tuning a FSFB controller, where a pole is defined for each state that is present in the model. This is a very popular, simple method of control as little knowledge of the overall system is required to attain a stable response. In this case we have four states (ball displacement, ball velocity, angular displacement and angular velocity). These poles are defined in a vector 'P' which is used with Ackermann's formula to determine the control gains of the system. MATLAB has a function which uses Matrix 'A' and 'B' along with the poles 'P' for Ackermann's formula. The MATLAB command for this can be seen below which accurately calculates these gains.

```
FSFB = acker(A,B,P);
```

Pole placement may take several attempts to reach a selection of poles which stabilise the system. A good starting point is to set all the poles to the same value. In the Figure 6 shown below it can be observed that the simulation stops at around half of a second as the ball falls off the beam and breaks the condition defined in the simulation. The conclusion that can be drawn from this is that the controller takes too long to respond to the changes in the balls position and so does not correct itself in time. To overcome this problem the pole must be pushed further towards the imaginary axis to get a faster response.

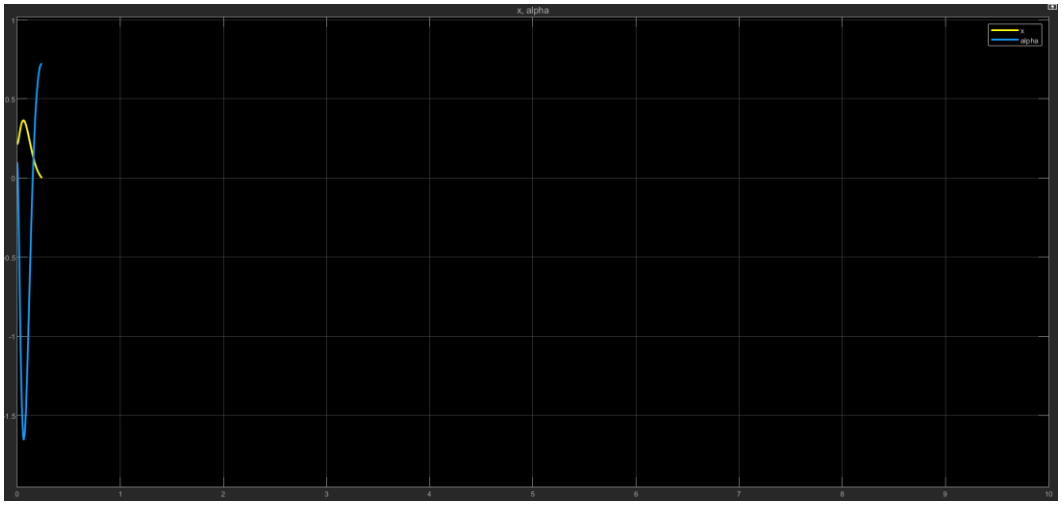

Figure 6 FSFB with four poles at -20, sim time = 10 s

The plot in Figure 7 shows the system behaviour with faster poles located at -3. As shown the poles provide a stable response and one which is much faster. As the simulation finishes the compensator catches the ball and stabilises without it falling off.

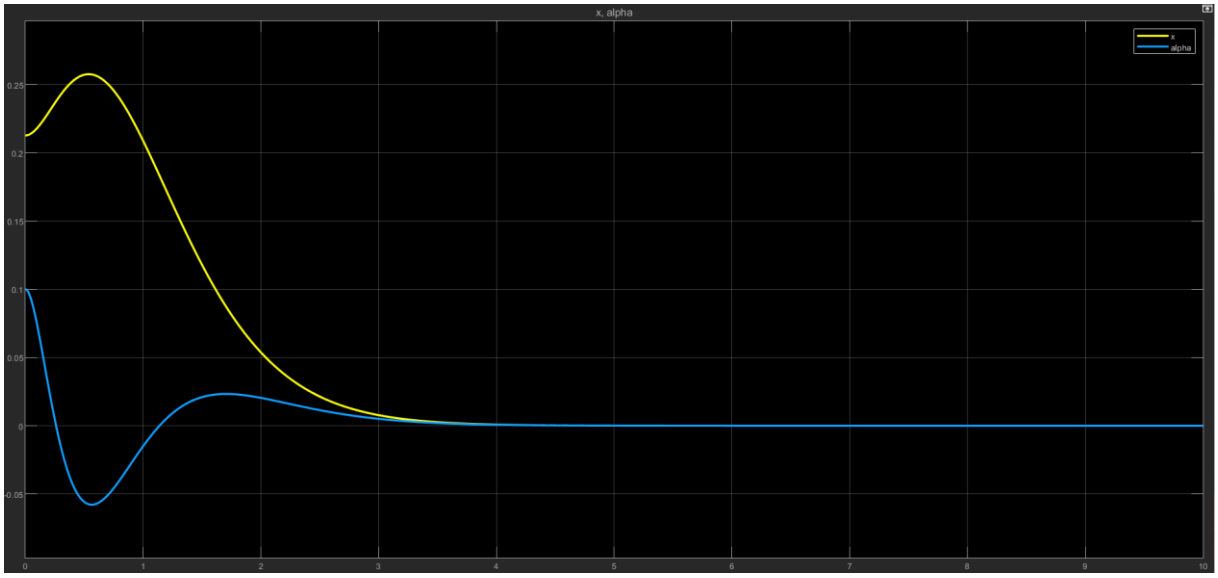

Figure 7 FSFB with four poles at -3

If this is approach is taken further an even faster response would be possible by selecting all poles as -1.5, as shown in Figure 8. However, in reality the system is now too underdamped for the controller to correct itself and the ball falls off.

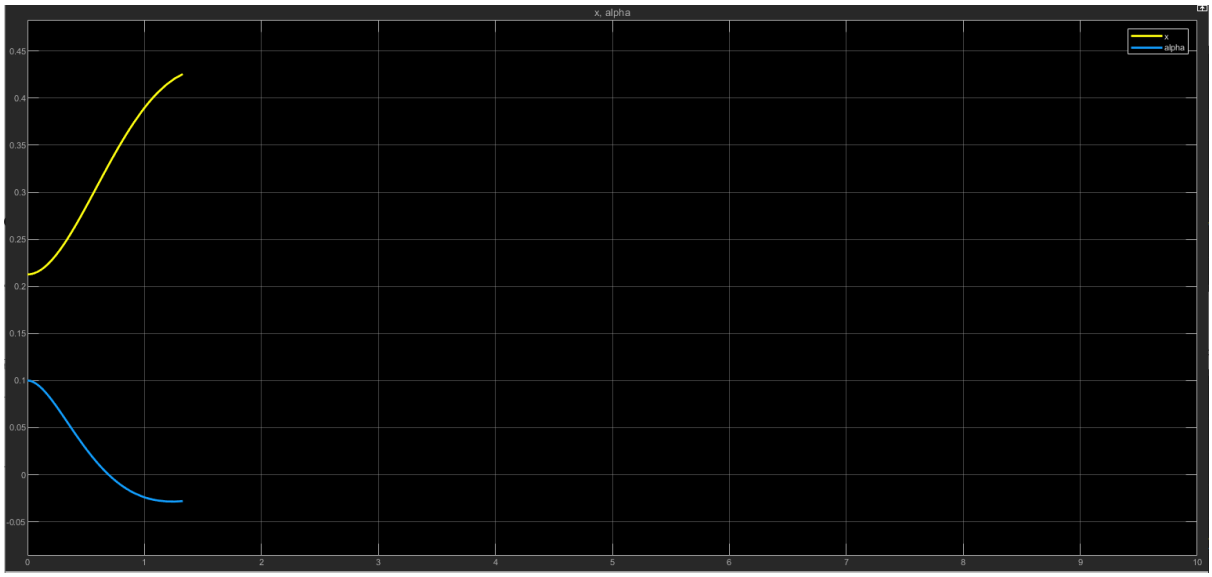

Figure 8: FSFB four poles at -1.5

Although, there is a solution, if two poles are left at -1.5 and two slower poles are moved back to -20 nearly the same speed of response is able to be achieved but with one which is stable and where the ball does not fall off the beam as seen in Figure 9 below.

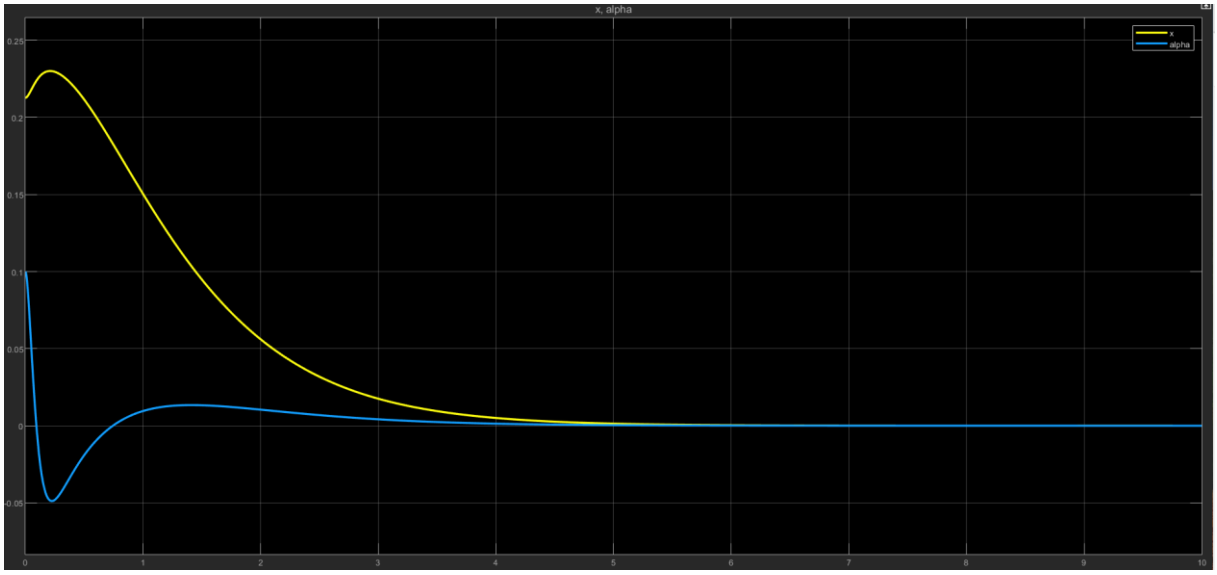

Figure 9 FSFB with poles at -20, -20, -1.5 and -1.5

Linear-Quadratic Regulator

Another popular method for selecting the Feedback gain (K) required for Full State Feedback is to use a Linear Quadratic Regulator (LQR). This is an Optimal Control Method and so guarantees the fastest response if tuned correctly. This method of tuning is more advanced as it requires a more in depth understanding of the system to be controlled and for what each state represents in the physical system. LQR is a method which uses a mathematical model to minimise a Cost Function by weighting

factors supplied by the engineer. These weighting factors are defined in a Q-Matrix, which is of length $n \times n$, where n is the number of states present in the state space. The Q-Matrix is configured by default a C'*C matrix, where values of x and α are defined as 1 and all others to zero. The diagonal elements from Q(1,1) to Q(4,4) represent the weighting of each state. It is for this reason that the engineer must understand the system and what each state represents so they are able to target areas for improvement.

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

Each weighting of the elements in the matrix affects the algorithm that influences the reduction of the cost function which results in a better fit. Table 1 below shows the definitions of each diagonal element of the Q-Matrix and to which State it has effect on.

Q(1,1)	Displacement x of the ball
Q(2,2)	Velocity $\dot{x}$ of the ball
Q(3,3)	Angle α of the beam
Q(4,4)	Angular $\dot{\alpha}$ velocity of the beam

Table 1 Meaning of elements in the Q matrix.

The value of R can be described as a symmetric positive definite matrix. It is to represent the R matrix which defines the weights on the control input, as there is a single input to this system there is no need for a matrix where a single value will work to the same accuracy. In this case R is set to be just 1.

During the tuning of the Q matrix only the elements of the diagonal were considered. All other values were set to zero because the elements of the diagonal only represent the effect of the corresponding state variable. Elements which are not on the diagonal represent the effect of two elements together (e.g. angular displacement with ball displacement together) on the design and considering these will be quite hard to comprehend these have been left as zero.

The Full State Feedback Gain K can be calculated by using the MATLAB command shown below:

```
FSFB = lqr(A,B,Q,R);
```

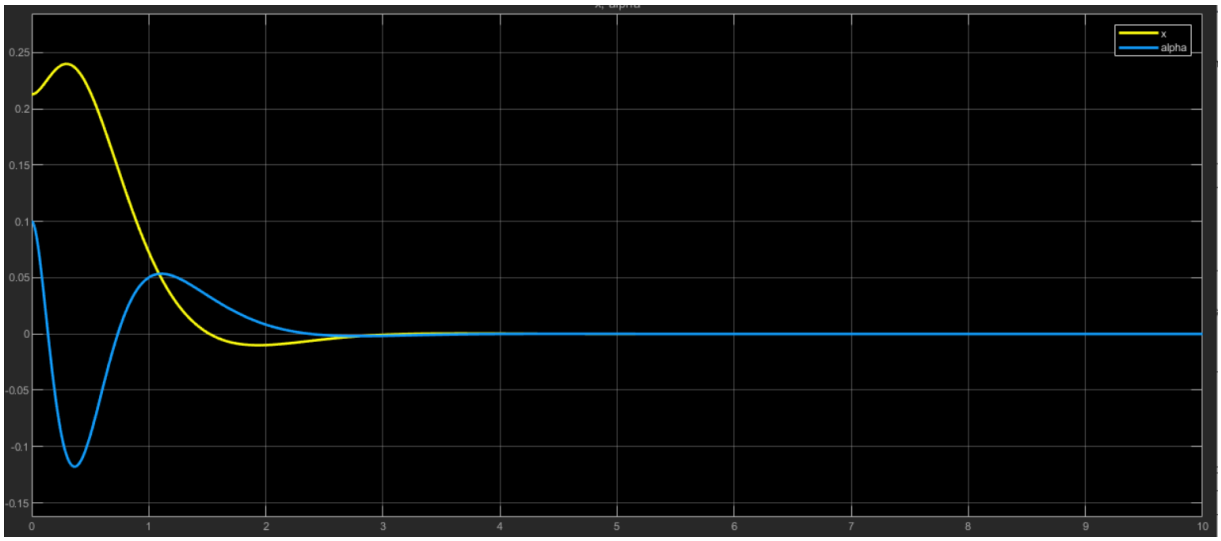

*Figure 10 FSFB with LQR design, Q Matric defined as C'*C.*

Figure 10 shows the basic system behaviour of LQR with a Q-Matrix using C'*C and the R matrix set to 1. It can be observed that the response of the system is stable as it converges to zero although does exceed the zero limit slightly. In order to determine appropriate values for each of the elements of the Q-matrix each parameter was tuned alone to a value of 500 to assess its relevance to the system. i.e., if a response fails with the element weighted out then it is of high priority for the system. As shown in Figure 11 and Figure 12, these responses resulted in the ball falling off the beam. The cost function is forced to zero either too fast, resulting in the ball flying off the beam or too slow resulting in an unresponsive controller allowing the ball to fall off slowly. In conclusion both values effecting the ball displacement and the angle of the beam should be kept low.

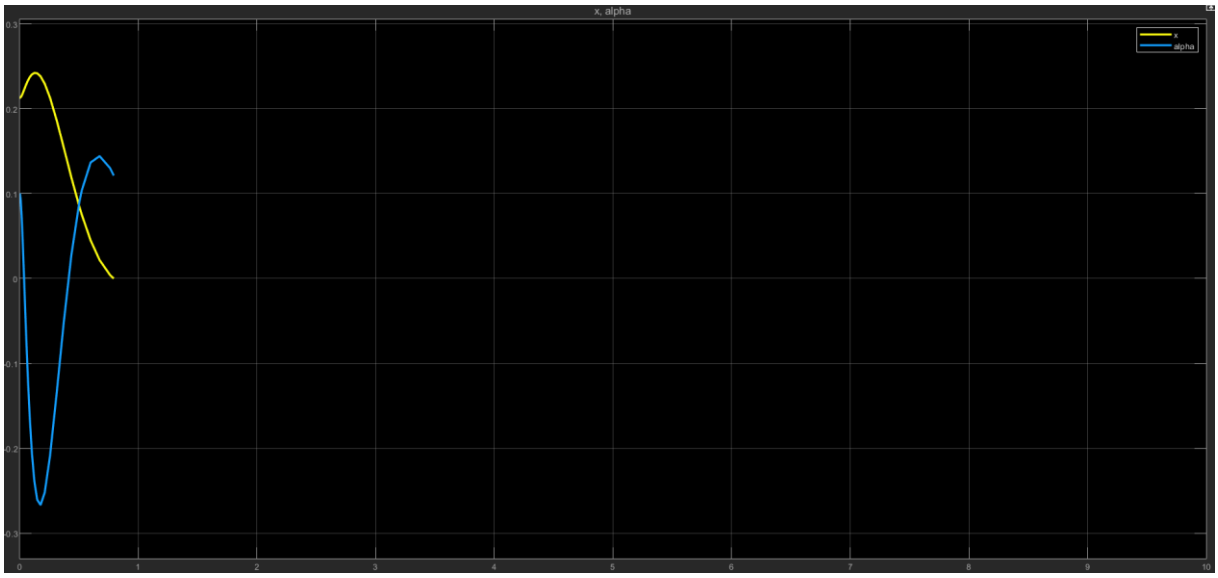

Figure 11 FSFB with LQR design, displacement weighted with 500.

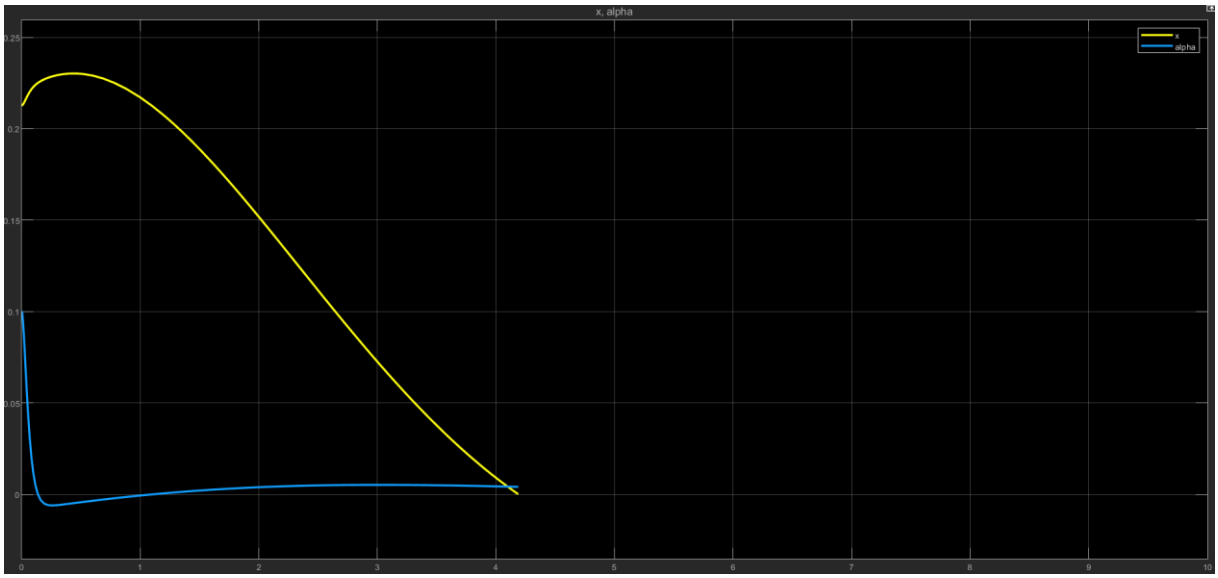

Figure 12 FSFB with LQR design, angle of the beam weighted with 500.

It was found however that the effect of increasing the value of elements Q(2,2) and Q(4,4) while having small values for Q(1,1) and Q(3,3) in general results in a good behaviour.

Using this method of trial and error, combined with knowledge of what each element of the matrix affects in the Q matrix, an accurate controller can be tuned with final values of the Q matrix shown below and the response shown in Figure 13. For our system this behaviour is fully sufficient.

$$Q = \begin{bmatrix} 62 & 0 & 0 & 0 \\ 0 & 250 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 150 \end{bmatrix}$$

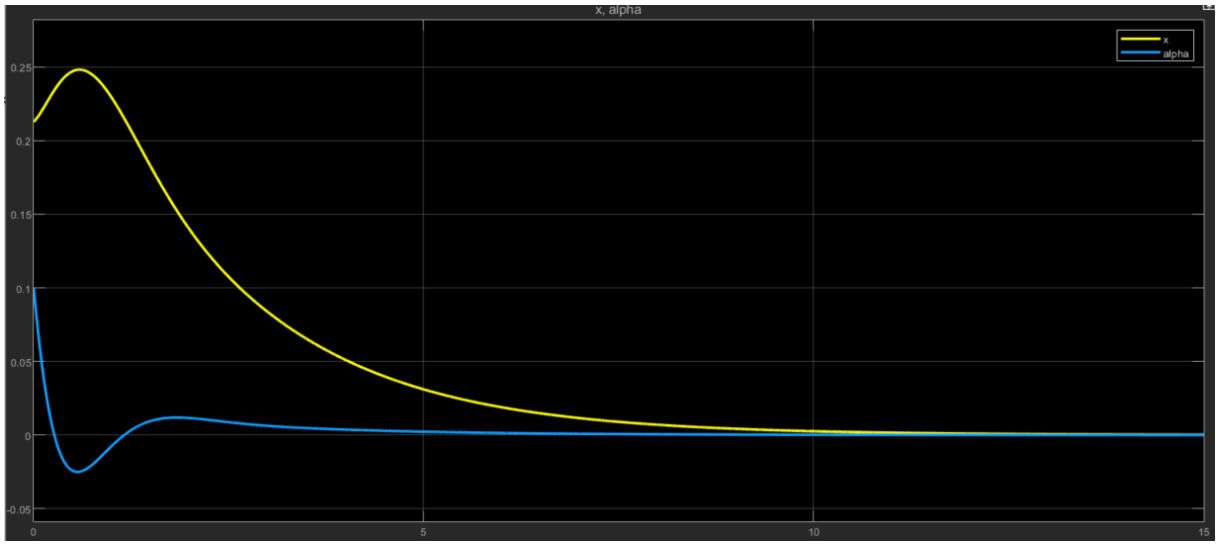

Figure 13 FSFB behaviour with LQR design, sim time = 15 s

Compensator Design

Whilst an acceptable system response is possible using Full State Feedback alone, it is computationally expensive on the controller as a constant feedback from sensors are required to achieve an accurate result. For this reason, a more robust controller can be assembled by using a compensator.

The compensator brings together the feedback gains calculated for the Full State Feedback controller but also makes use of a Luenberger Observer to estimate the values of the states of the system so less emphasis is put on highly accurate, constant sensor data.

The model of the Compensator combining both the Observer and the Full State Feedback gains can be seen in Figure 14 below. The main change from the Full State Feedback design seen in Figure 5 is that the input of the State Feedback is taken not from the states 'x' but from the new Observer estimated state '$\hat{x}$' .

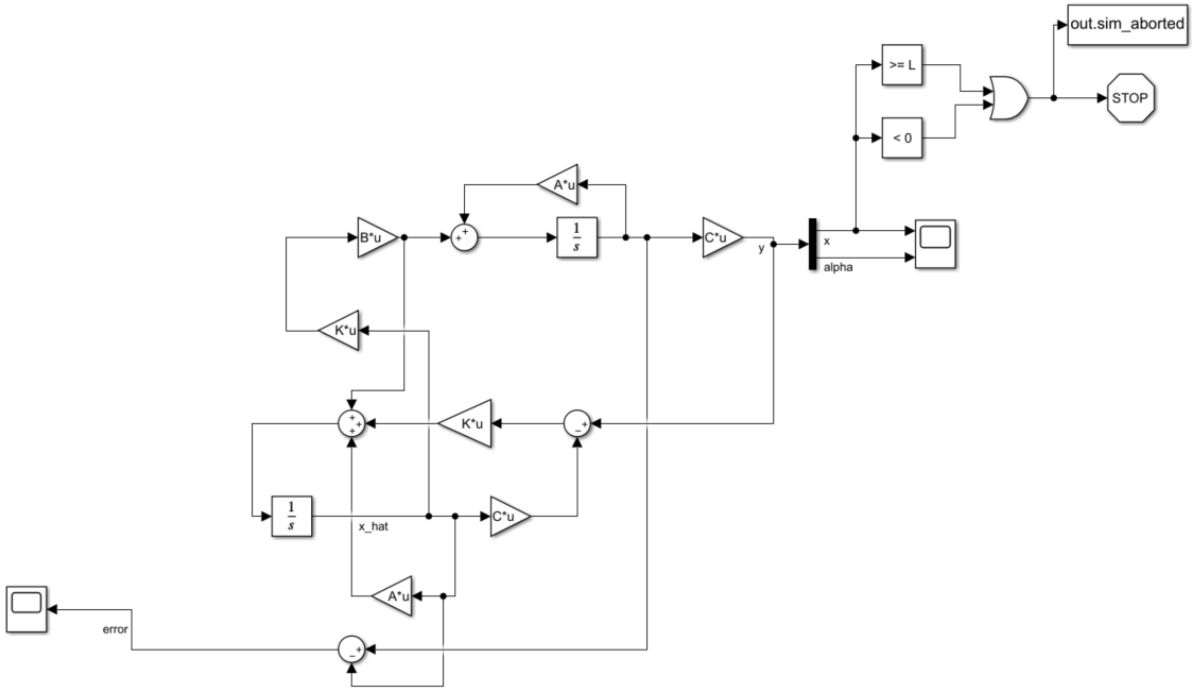

Figure 14 Simulink model of compensator design

To use the Luenberger Observer first a series of Observer gains must be calculated. Generally, the Observer poles have a direct association with the poles selected as part of the Pole Placement process used in Full State Feedback. The poles of the Observer are usually calculated as an integer multiple of these Full State Feedback Poles. In general, the Observer poles range from three to ten times the value of the Full State Feedback poles. The higher the values of these Observer poles, the faster the observer eliminates an error to zero. This can be seen in Figure 15 and Figure 16. Both figures look graphically very similar however the scaling is entirely different. Figure 14 where the Observer poles are three times larger than the Full State Feedback poles settles to zero error in around 1 second. However, in Figure 15 with Observer poles of ten times the value of Full State Feedback poles, the elimination of error to zero occurs in around 0.3 seconds.

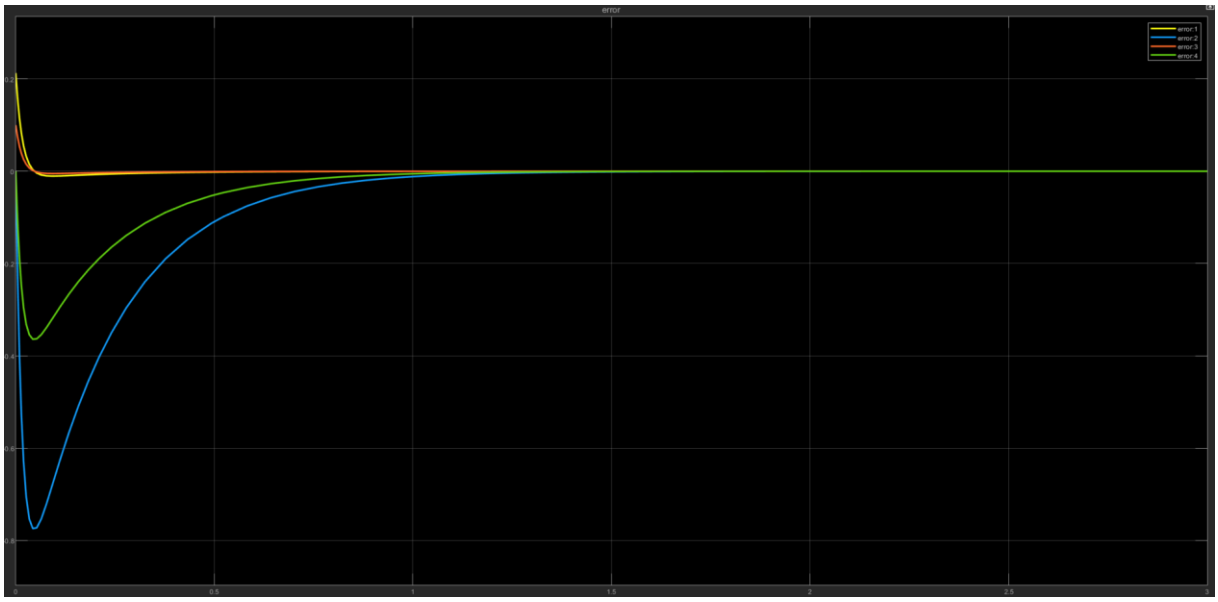

Figure 15 Estimation error with observer poles three times the FSFB poles, sim time = 3 sec

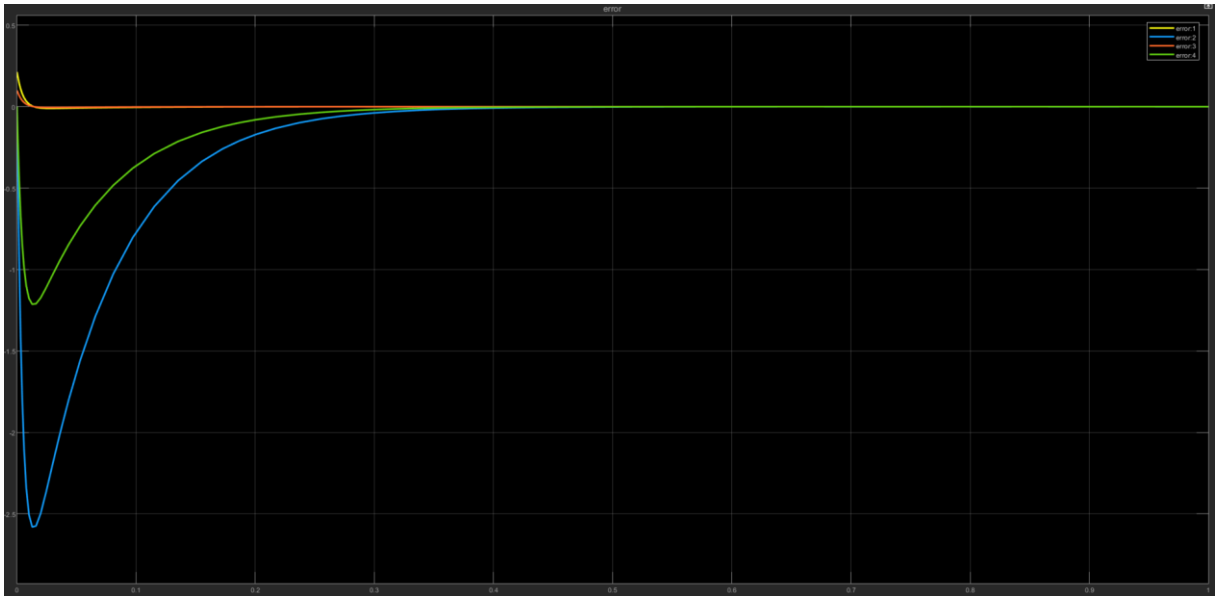

Figure 16 Estimation error with observer poles ten times the FSFB poles, sim time = 1 sec

The initial conditions of the $\hat{x}$ integrator are [0.1;0;0;0] i.e., the estimation of the displacement is the only non-zero value. Usually there is no better guess for any state variable than zero, however, in the case of ball displacement, it is logical to set this to a non-zero value as the simulation is set to stop when the displacement is out of the range 0 to length of the beam. Nevertheless, we tried to be as close to zero as we could get and 0.1 as a starting point for the estimation works to a satisfactory level.

Pole Placement

For the pole placement in the compensator the poles -20, -20, -1.5 and -1.5 were used. In Figure 17 an observer with 3 times the poles of the full state feedback were used. It can be observed that as the value of x dips below 0 slightly the ball falls of the beam. This is because the observer is too slow in estimating the values of the state variables. To compensate that in Figure 18 the observer poles were calculated as 10 times the full state feedback poles the result of this being, the ball does not fall of the beam and stabilises in around 2.5 seconds.

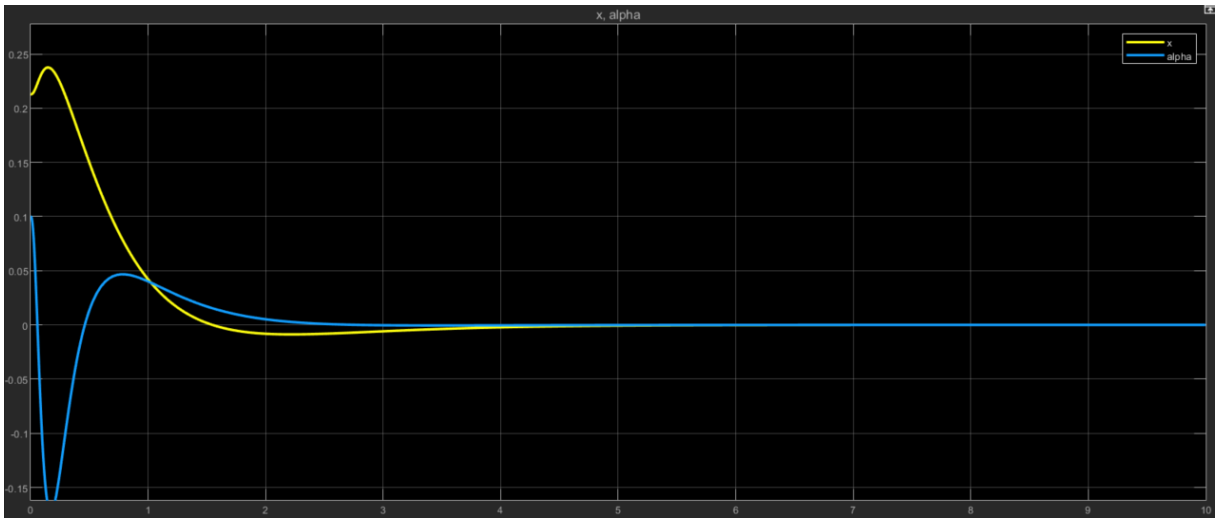

Figure 17 Pole placement in the compensator with too slow observer, sim time = 10 s

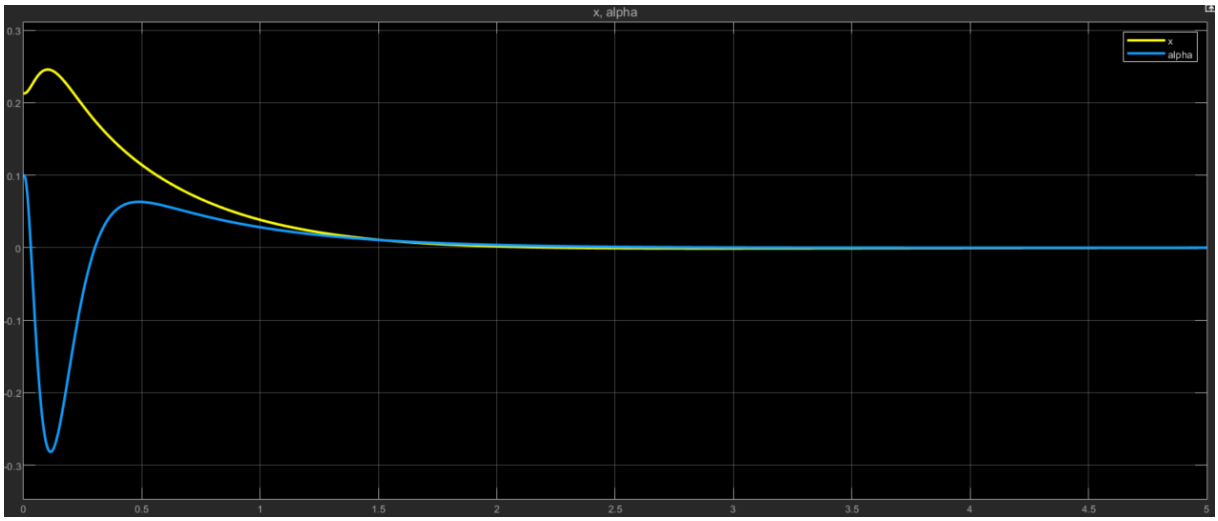

Figure 18 Working pole placement in the compensator, sim time = 5s

LQR Design

The observer poles should generally be three to ten times the full state feedback poles but in the LQR design of the full state feedback you do not specify any poles. Because of this you first have to calculate the poles of the full state feedback system and use them for calculating the observer gain L like:

```
full_state_poles = eig(A-B*K)
Lo = place(A', C', full_state_poles .* 10)';
```

Using ten times the poles of the full state feedback leads to a compensator behaviour that is shown in Figure 19.

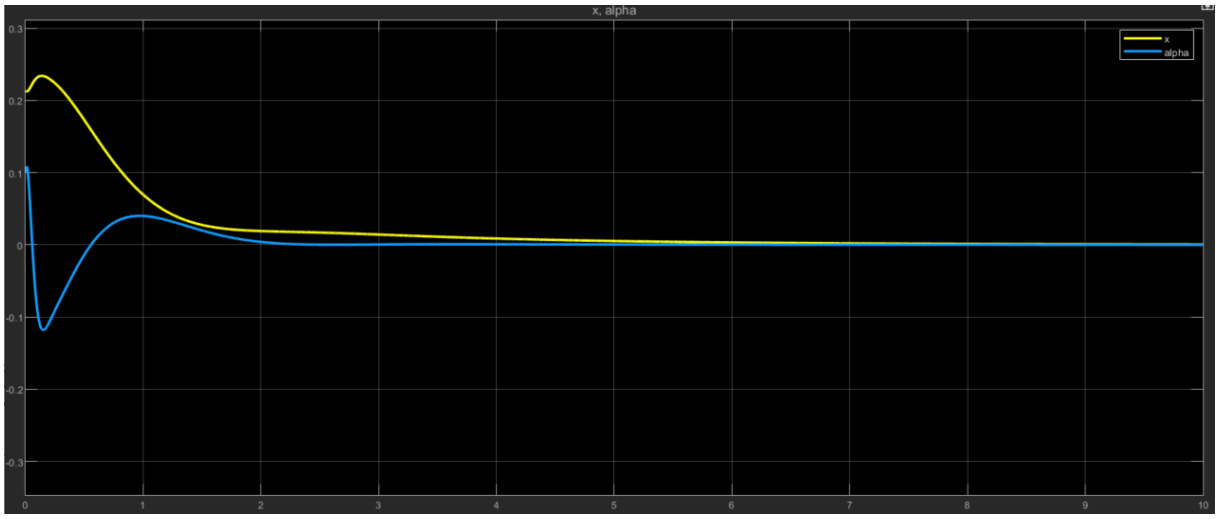

Figure 19 Working compensator with LQR design, sim time = 10s

Test Compensator Design in Nonlinear Model

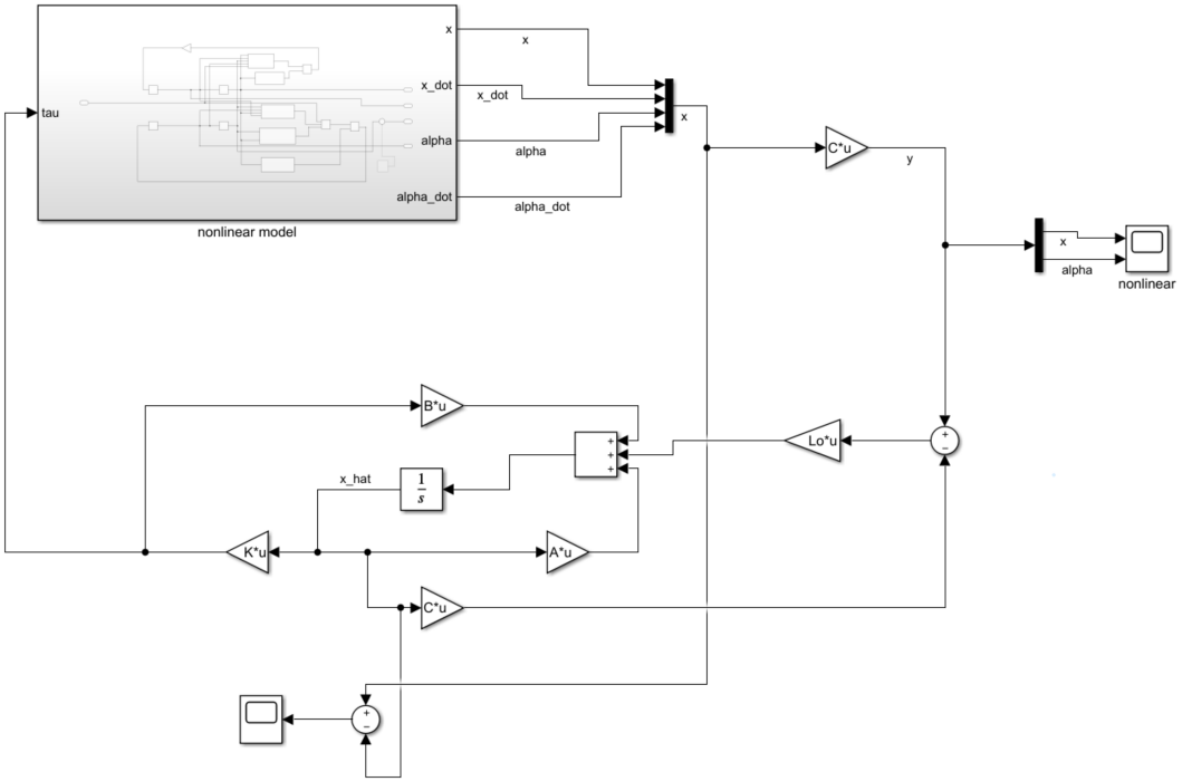

Figure 20 Compensator design connected with the nonlinear model

After the compensator is designed with the Linear model the next step is to test this compensator design with the Nonlinear model. For that the states of the Nonlinear model must be extracted and connected with to a multiplexer to attain the state vector x that can be used as in the Linear model. With this vector the matrices that were used in the Linearized model can be used in the Nonlinear model. The output of the compensator is then fed back into the Nonlinear model as the time constant τ as shown in Figure 20. In general, it is easier to get the Nonlinear working with the LQR designed compensator because the LQR already contains an optimization method. The behaviour of the Nonlinear model can be observed form Figure 21. The displacement does not stabilize at zero but as

the angle of the beam is settled at zero the overall system is stabilized. Additionally, we do not violate the physical limits of the beam i.e., that the ball does not fall of any edge.

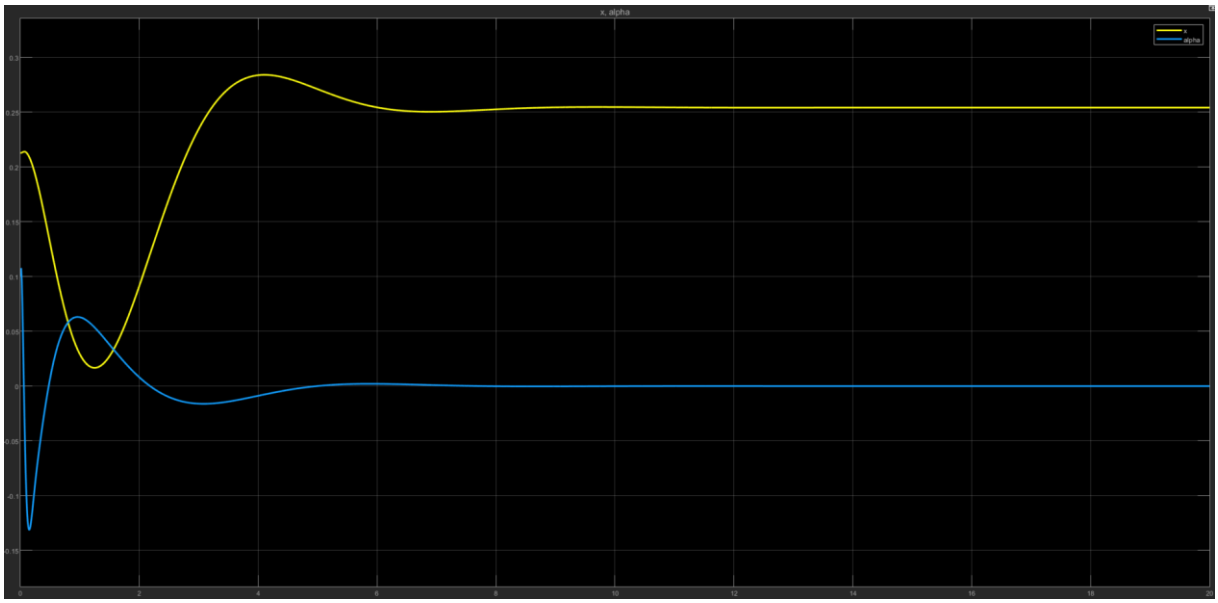

Figure 21 Nonlinear model controlled with the LQR designed compensator design, sim time = 20s

Sensor and Actuator Errors

After the general functionality of the compensator is tested in a Nonlinear model it is a good idea to see how the system reacts on some Gaussian noise that can be added to the sensors of the system as shown in Figure 22, or in the actuator as shown in Figure 23. Simulink provides the Random source block for this purpose to get Gaussian noise and add it to a particular signal. The values for the errors should not be too extreme or it is not realistic that the system will be able to compensate and stabilise such an error.

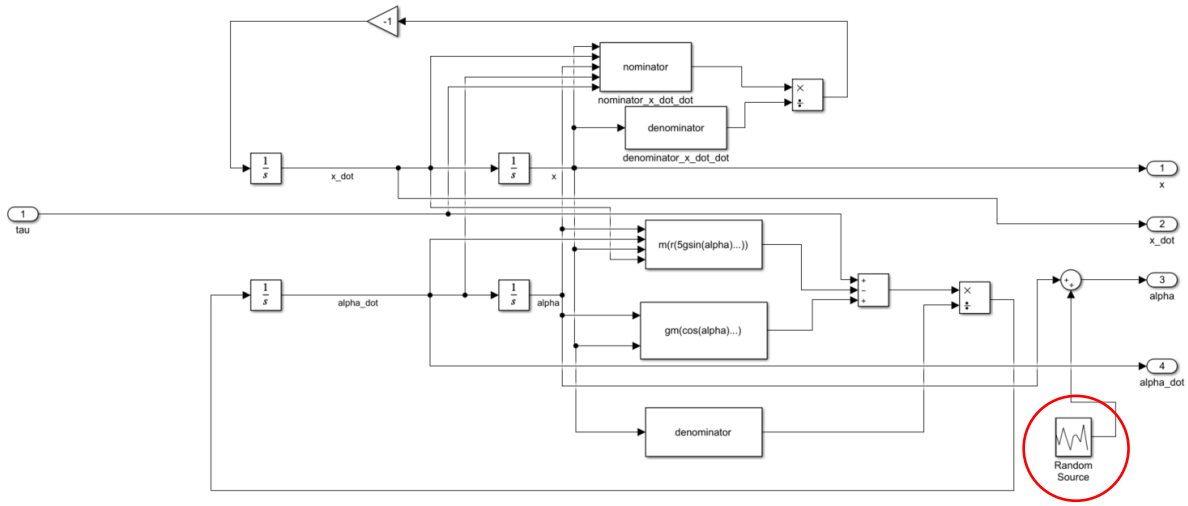

Figure 22 Actuator error in the nonlinear model

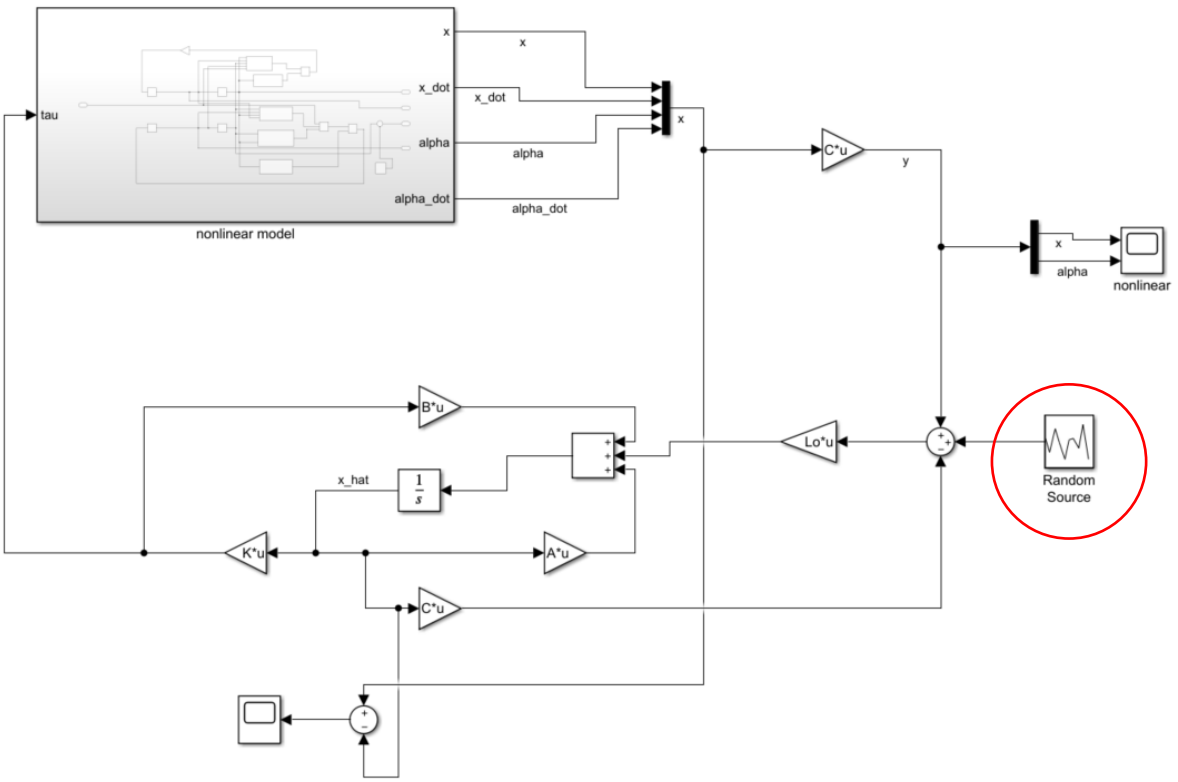

Figure 23 Sensor error in the nonlinear model

In Figure 24 the behaviour of the compensator design in the nonlinear model with a sensor error added can be observed as for the model Figure 23. The sensor error was limited to ± 0.01 m for the displacement and ± 0.01 radian for the angle.

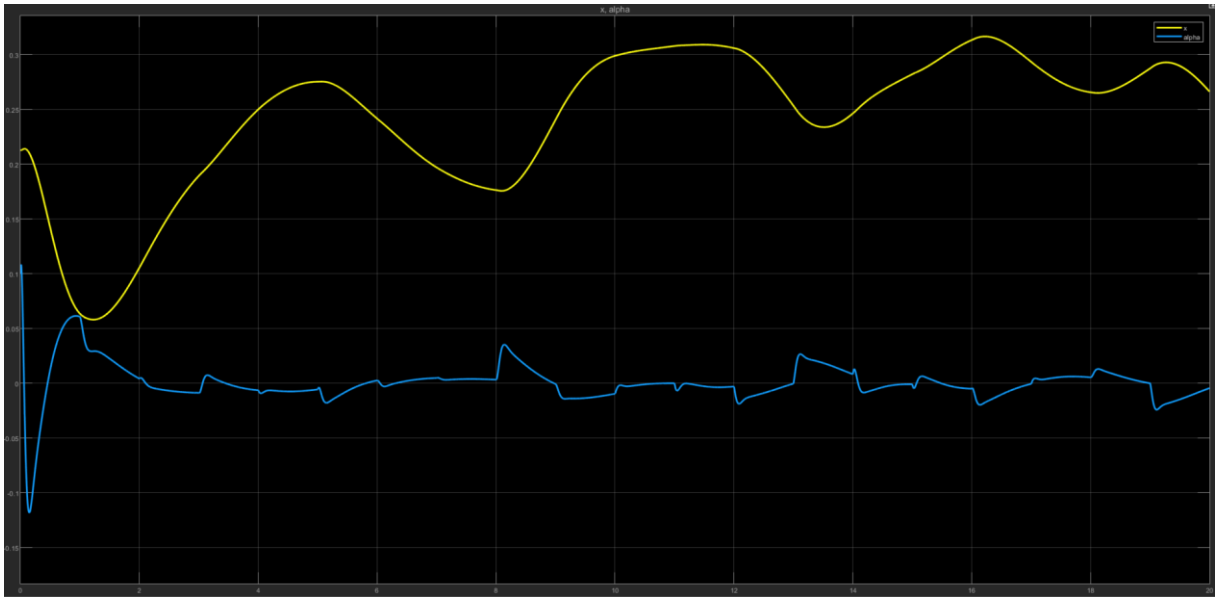

Figure 24 Nonlinear model with sensor errors, sim time = 20s

In Figure 25 the effect of an angle actuator error is shown when added to the model shown in Figure 22 with a minimum of -0.01 radian and a maximum of 0.02 radian.

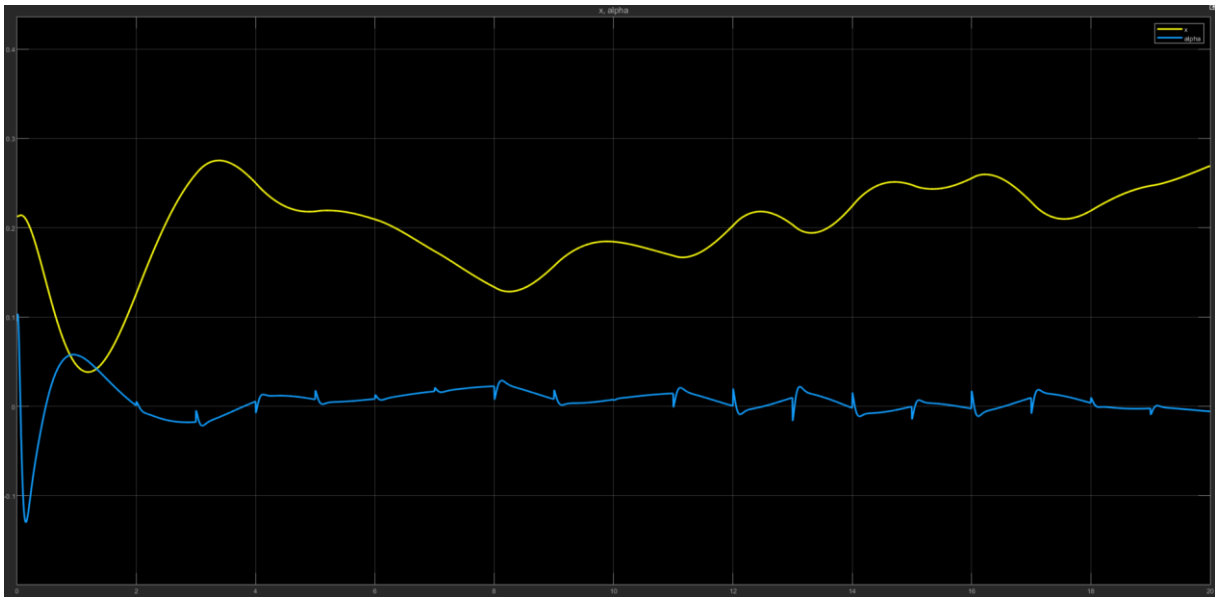

Figure 25 Nonlinear model with angle actuator error, sim time = 20s

It can be observed that for both Figure 24 and Figure 25 that our compensator design is able to compensate this error. Although it would appear the final value of x is never stationary it only moves between the limits of 0 and L and so consequently never falls off the beam.

Conclusion

This project started with an unstable Nonlinear system representing a ball beam that should be controlled. After linearizing the system, it was analysed by assessing the values of the open loop poles and if the system is able to be controlled and/or observed. A controller was then designed for the system with two design methods for the full state feedback and compensator. A PID control method was also explored however it was not further pursued in this task. With Pole Placement and the LQR design it was possible to control the linearized system. The LQR design was also used to test the compensator on the nonlinear model and even with sensor and actuator errors the ball remained stable on the beam.