

# UNIVERSITY OF CAPE TOWN

## Department of Electrical Engineering



### **EEE4118F – Process Control and Instrumentation Project 2019 Report**

**Tareeq Saley**

**17/04/2019**

## Executive Summary

In order to achieve the aim of designing, implementing and evaluating a digital controller for the D.C. Motor, the following steps were taken:

- The required specifications of the controller were formulated
- A thorough understanding of the system was developed and all its limitations were understood
- The system was broken down into subsystems and the properties of each were considered
- Step and ramp tests were conducted in order to determine the internal parameters of the system
- This understanding of the system and its internal parameters were then used to develop a mathematical model that was used in the design and development of suitable controllers
- Two control methods were explored and developed, one being a state feedback controller, the other being a model predictive controller
- Both controllers were then simulated through Matlab or Scilab, and thereafter implemented in C Sharp in order for the system response to be evaluated and compared with the simulated and expected results.
- These controllers were then compared with each other with regards to them meeting specifications set out in the project brief.

In essence, the project served to provide an understanding as to the differences between classical control design techniques such as state feedback control, and modern control design techniques such as model predictive control. One would think that modern control design techniques far surpass the classical control approach, especially in achieving and possibly surpassing specific technical specifications, however, that wasn't necessarily the case in this project.

Both controllers greatly improved the closed loop system performance with a fast and non-oscillatory response. The controllers were also able to reject input disturbances and were not sensitive to process changes. The ease in which these controllers as well as the non-linear elements could be implemented digitally was also very important.

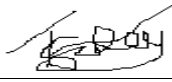
The fact that the tachometer (motor speed) as well the output position were both taken in as inputs improved the ability of the controller to perform the task of position control and rendered the gearing ratio obsolete. This was due to the fact that instead of taking the derivative of the position in order to obtain the speed, the real speed was inputted directly.

The results that were obtained during the implementation of the controller with the D.C. Motor were in line with the results that were simulated for the state feedback controller as well as the design specifications that were initially developed.

## Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this final year project report from the work(s) of other people, has been attributed and has been cited and referenced.
3. This final year project report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Name: Tareeq Saley

Signature: 

Date: 04/17/2019

## Contents

Executive Summary .....	2
Declaration .....	3
Introduction and Project Plan .....	5
Problem Formulation .....	6
Component Blocks .....	6
Op Amp and Power Amplifier Blocks .....	6
Non-linearity Block .....	6
DC Motor Block: .....	7
Block Analysis of DC Servo Motor .....	7
System Identification and Mathematical Modelling .....	8
Mathematical Modelling .....	8
Dynamic Modelling and Model Verification .....	10
System Structure .....	11
State Feedback Controller Design .....	11
Controller Selection .....	11
Controller Design .....	12
Controller Simulation .....	13
State Feedback Controller Implementation .....	14
Conversion Into the z-Domain .....	14
State Feedback Controller Experimental Results .....	15
MPC Controller Design .....	15
Controller Design .....	15
Controller Simulation .....	17
MPC Controller Implementation and Results .....	18
MPC Experimental Results .....	18
Controller Comparison and Conclusion .....	19
Simulated Response .....	19
Performance Criterion .....	19
MPC Controller .....	19
Implemented Response .....	19
References .....	20
Appendix A – Classical Control Design .....	21
.....	23
Appendix B – MPC Controller Design .....	24

## Introduction and Project Plan

### Motivation

A control system is responsible for controlling the various types of process variables in industry or any plants. Control systems are important because they help develop robust, optimal solutions to problems, whilst ensuring that those solutions manage to take care of unexpected changes, noise or disturbances to a particular system.

### Aim

The aim of this project is to design, implement and evaluate a digital control system for position control of a non-linear D.C. Motor.

### Requirements

The system will be expected to provide closed loop performance that satisfies the following requirements:

- The response is expected to be stable
- The response should track position set points with zero error
- The speed of response should be faster than the response time of the uncontrolled plant
- The response should be sufficiently damped
- The response should reject input disturbances
- The response should have less than 20% overshoot in rejecting disturbances
- The system must operate within the voltage range of  $\pm 10\text{V DC}$
- The system must be able to handle the non-linearities associated with the plant

### Project Plan

In order to design a controller for the D.C. Motor that satisfies these requirements, the motor and all its subsystems need to be carefully analysed and understood and a suitable process model for the system needs to be developed. This model will then be used in the design of the controller.

The process of designing a suitable controller for the servomotor system involves having to gather experimental data from the uncontrolled system, and having to use that data together with mathematical analysis techniques and system identification techniques to find an expression for the controller to be designed.

The controller is then designed using two different design methods, the state feedback control method and the model predictive control method. The controller is then fine-tuned and further developed for both methods, by considering the limitations of the system and the criterion that the controller is expected to satisfy.

It is then simulated and implemented on the real-life DC motor system in order to gauge its response and draw comparisons relative to the theoretical expectations set out for it, and to evaluate whether the developed controller performs as intended and meets its requirements. The two controller design methods will then be compared with each other to determine which method suited this project best.

By the 13<sup>th</sup> of April 2019, all experiments are meant to be completed, with the project report to be submitted by the 17<sup>th</sup> of April 2019.

## Problem Formulation

The system identification method is used to design a model of a dynamic, linear and nonlinear system which are based on measured data. The DC motor system from which the data was obtained is a closed loop, feedback system and is represented by the following block diagram:

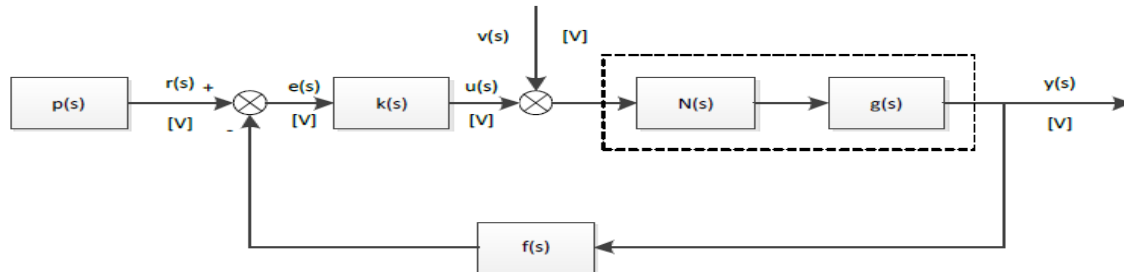


Figure 1: Closed loop system block diagram

It should be noted that the process model consists of  $G(s)$  and  $N(s)$  which represents the linear plant model and the non-linearities that are associated with it respectively. A concise open-loop block diagram of the system related to its physical components is given below:

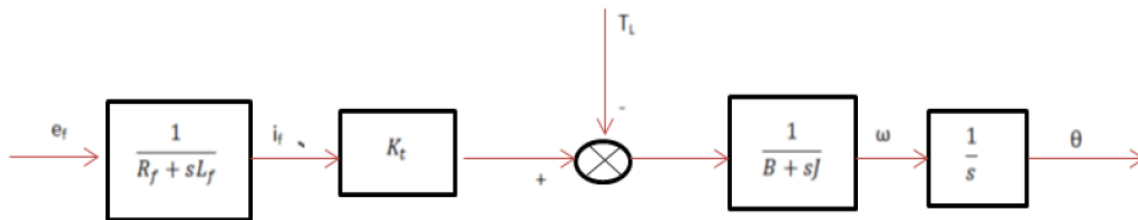


Figure 2: Open loop block diagram of DC motor system (Mohan, 2002)

### Component Blocks

An expanded open loop block diagram of the plant model  $G(s)$  is given below. There is, however, a non-linearity component that also influences the system that will be discussed below.

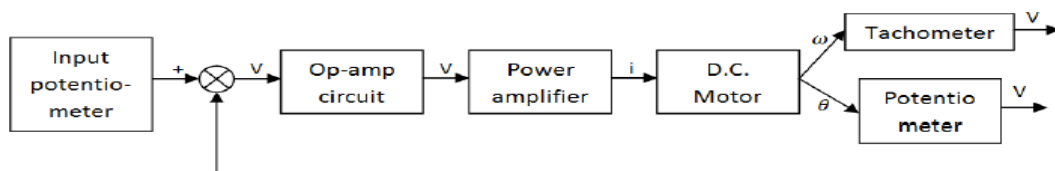


Figure 3: Expanded open loop block diagram

### Op Amp and Power Amplifier Blocks

The op amp serves as a high impedance buffer and provides differential signal operations. It inverts the signal coming from the controller.

### Non-linearity Block

There are two non-linear elements present within the motor; deadband and saturation. The deadband is due to the input voltage not creating a moment large enough combat the effects of inertia and frictional torque exerted by the motor on itself (stiction). This prevents the motor from moving. Saturation is due to the motor being constrained by potentiometer limitations. To allow for the system to be modelled adequately, the area within which the motor starts to move will be assumed to be linear. The non-linear elements discussed above will be taken care of when coding the controller before implementation. The effect of these non-linearities on the motor speed is illustrated below:

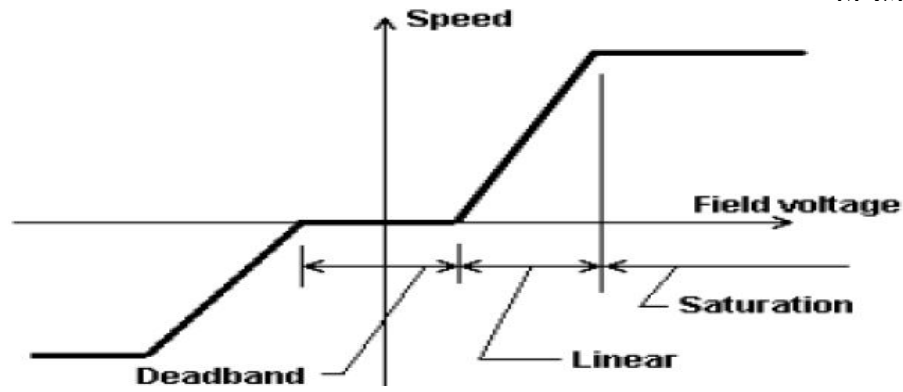


Figure 4: Motor speed vs voltage in the presence of non-linearities

#### DC Motor Block:

As can be seen in figure 5 below, an armature voltage, which controls the DC motor, develops an armature current through the resistor, inductor and load. This creates a voltage across the motor which results in a torque that is proportional to the armature current. Other factors of the motor that need to be considered are its internal inertial mass and damping factor.

#### Block Analysis of DC Servo Motor

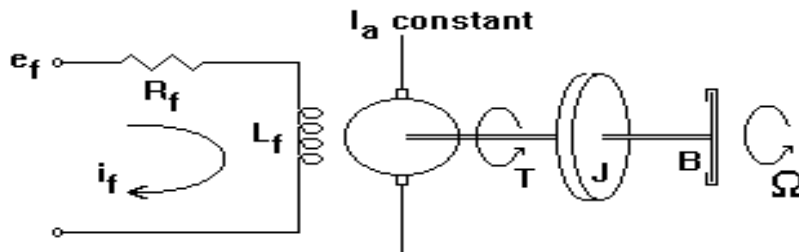


Figure 5: Field Controlled DC Motor

Assuming constant armature current, the torque developed by the motor is given by

$$T(s) = K_t i_f(s)$$

The field coil currents can be represented as

$$i_f(s) = \frac{1}{R_f + sL_f} e_f(s)$$

The torque developed by the magnetic fields is balanced against the mechanical damping and inertia so that

$$T(s) = \{sB + s^2J\}\theta(s)$$

By combining the above three equations, and after further simplification, the position of the output shaft,  $\theta(s)$ , is related to the field voltage  $e_f(s)$  by

$$\frac{\theta(s)}{e_f(s)} = \frac{K}{s\{1 + sT_m\}\{1 + sT_f\}}$$

With  $K = \frac{K_t}{BR_f}$  being the motor gain in  $[Volt - seconds]^{-1}$

$T_m = \frac{J}{B}$  being the motors mechanical time constant in  $[seconds]$

$T_f = \frac{L_f}{R_f}$  being the motors field time constant in  $[seconds]$

Since  $T_f \ll T_m$ , the third order transfer function can be simplified and approximated as a second order transfer function

$$\frac{\theta(s)}{e_f(s)} = \frac{K}{s\{1 + sT_m\}}$$

Finally, this transfer function is differentiated to relate the field voltage to the angular velocity of the output shaft instead of its position

$$\frac{\omega(s)}{e_f(s)} = \frac{s\theta(s)}{e_f(s)} = \frac{K}{\{1 + sT_m\}}$$

This results in the transfer function representing a first order system. (Braae, 2011)

## System Identification and Mathematical Modelling

Using the respective attenuation value of 3 that was given, the ramp response of the DC motor was tested. Close attention was paid to finding the linear and nonlinear regions between the deadband and saturation areas and only the linear regions were used for further analysis and step response data capturing using control theory methods. The non-linearities would be accounted for at a later stage in this project using model predictive control.

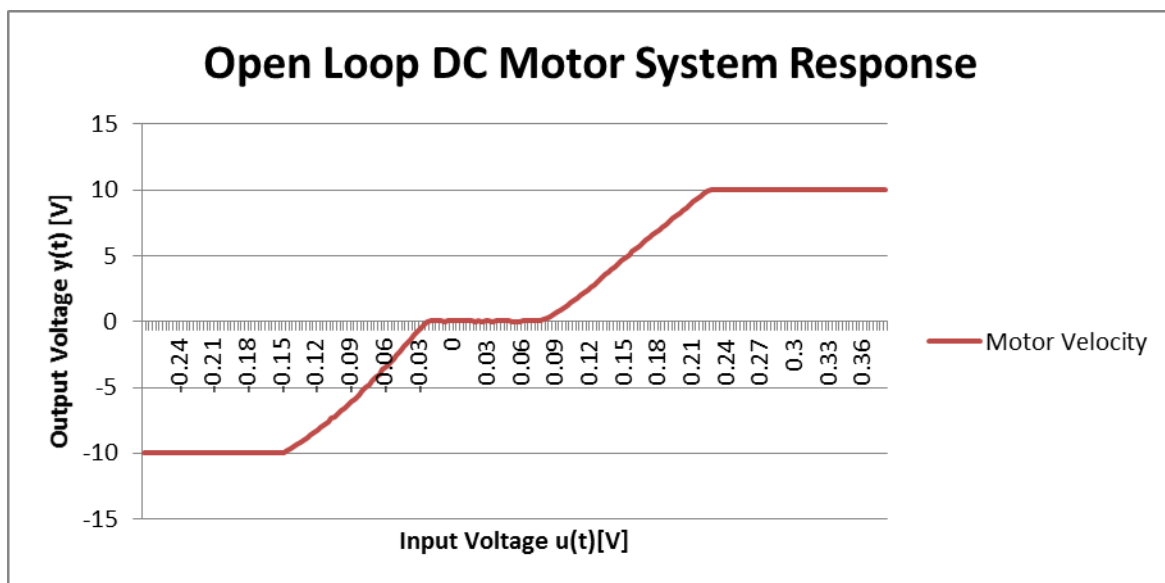


Figure 6: Open Loop DC Motor System Response

As can be seen in figure 6, the linear/nonlinear regions of the system's response ranges between  $[0.09, 0.21]$  and  $[-0.03, -0.13]$  and stepping to values out of these regions would lead the system into saturation or into the deadband between those regions.

### Mathematical Modelling

The step response method was used initially to determine the order of the system in question. The response matched up to a first order system, the standard form of a first order transfer function which is to be modeled is,

$$r(s) = \frac{A}{1 + Ts}$$

Where  $T$  represents the time constant value of the graph and  $A$  represents the steady state gain of the graph. This transfer function represents the angular velocity transfer function of the system with respect to the input. To find the output of the system relating angular position to the input, we update the transfer function as follows:

$$G(s) = \frac{AB}{(1 + Ts)s}$$

Four sets of data were collected, a set for both positive and negative step responses in both the positive and negative linear regions of the curve representing the actual, physical system. The



maximum velocity value (albeit in volts) from each resulting step response curve could then be used to find A and then T respectively. This is because, the steady state gain of the transfer function is given by AB, this comes from the final value theorem. The formulae used to calculate A and T respectively are given below:

$$A = \frac{\text{Final Value} - \text{Initial Value}}{B}$$

With B being the value of the step, and the final and initial values corresponding to the velocities before and after the step was implemented.

$$V = (\text{Final Value} - \text{Initial Value}) * 63.2\% + \text{Initial Value}$$

Using excel, the calculated value 'V' was compared to the motor velocity data to find the closest matching data points. The 15 surrounding data values that were nearest to the value V were used with their respective time values to formulate a straight line equation and linear interpolation was used to then derive a matching T (time constant) value.

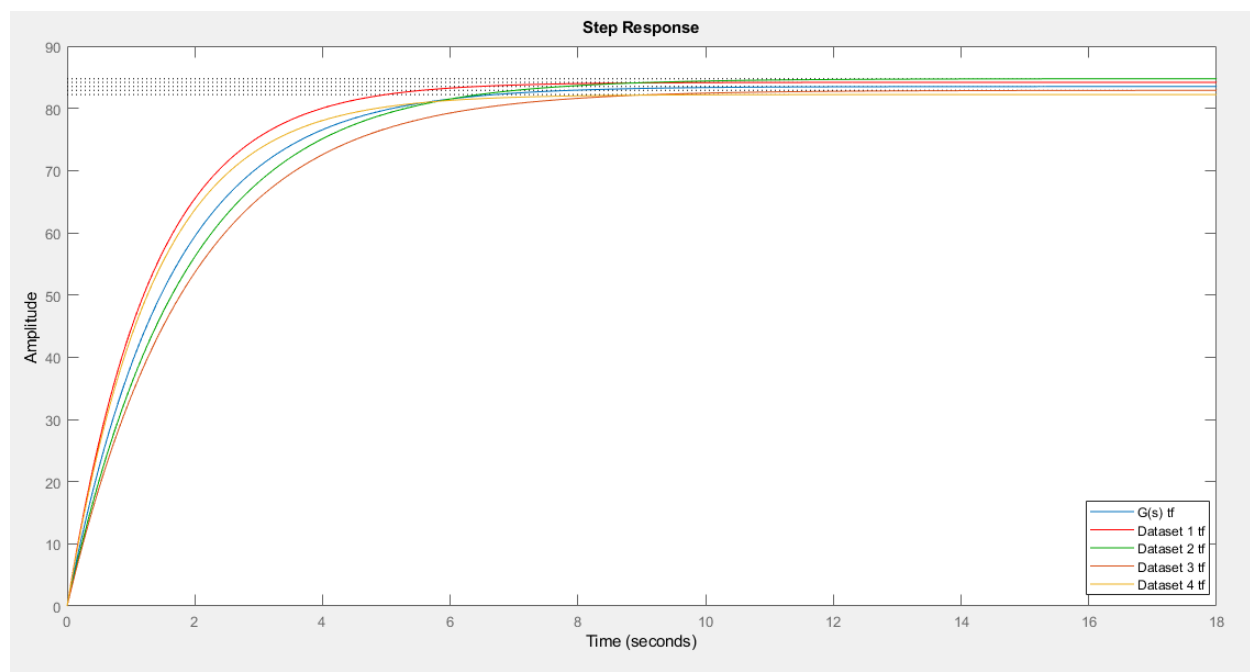
The initial value is used as the point of reference and thus should be added in the above formula to produce a time constant relative to it and not to the 0 point on the axis. The corresponding A and T values for each respective step is given below:

Region	Step	A (V)	T (s)
Positive Linear	0.05	84.18	1.3289
Positive Linear	-0.05	84.76	1.8406
Negative Linear	0.05	82.90	1.9213
Negative Linear	-0.05	82.22	1.3418

The average A and T values based on the above table give A = 83.515V and T = 1.6082s. Therefore, the calculated first order plant transfer function relating to the servomotor system's angular velocity is:

$$r(s) = \frac{83.515}{1 + 1.6082s}$$

In order to analyze the accuracy of this transfer function compared to the four data sets that were used to develop it, the step response of this transfer function was plotted against the step responses of each of the other four curves resulting in the following:



It can be seen that  $G(s)$  clearly serves as a ‘best fit’ curve and fairly serves as an average curve to represent all of the response datasets. The response of  $G(s)$  maintains at least a 98.44% accuracy rate and its transfer function can therefore be used for further analysis. Also, it’s evident that the simulated response for each dataset does not take into account real life discrepancies that caused the physical systems response to oscillate and the dead time between linear regions of operation. These offsets will have to be compensated for when designing the controller.

To find the output of the system we update the transfer function to relate angular position to input voltage in order to track position of a type 1 system. Therefore, the output transfer function is:

$$G(s) = \frac{83.515}{(1 + 1.6082s)s} \frac{[\theta]}{[V]}$$

However, the mechanical coupling between the motor and potentiometer needs to be accounted for in order to control the output potentiometer position. The gearing ratio is given as [1:30]. Therefore:

$$G(s) = \frac{1}{30} \frac{83.515 \times 0.05}{(1 + 1.6082s)s} = \frac{1}{30} \frac{83.515}{(1 + 1.6082s)s} = \frac{2.784}{(1 + 1.6082s)s} \frac{[V]}{[V]}$$

## Dynamic Modelling and Model Verification

To ensure robustness and accuracy in controller design, the transfer function obtained above needs to be verified to ensure that the controller to be designed will be based off an accurate system model.

The initial and final value theorems were used in order to verify the mathematical model  $g(s)$  that was developed.

FVT:

$$y(\infty) = \lim_{s \rightarrow 0} s y(s) = \lim_{s \rightarrow 0} s \frac{BA}{s(1 + Ts)} = BA$$

IVT:

$$y(0) = \lim_{s \rightarrow \infty} s y(s) = \lim_{s \rightarrow \infty} s \frac{BA}{s(1 + Ts)} = 0$$

The values of IVT and FVT coincide with the mathematical model that was developed above.

The root locus technique was then used to determine the stability concerns, if any, of the system and whether a controller is needed to rectify it.

The characteristic equation  $\phi_c$  is given by:

$$\phi_c = s(1 + 1.6082s)$$

The open-loop poles are therefore at  $s = 0$  and  $s = -0.6218$

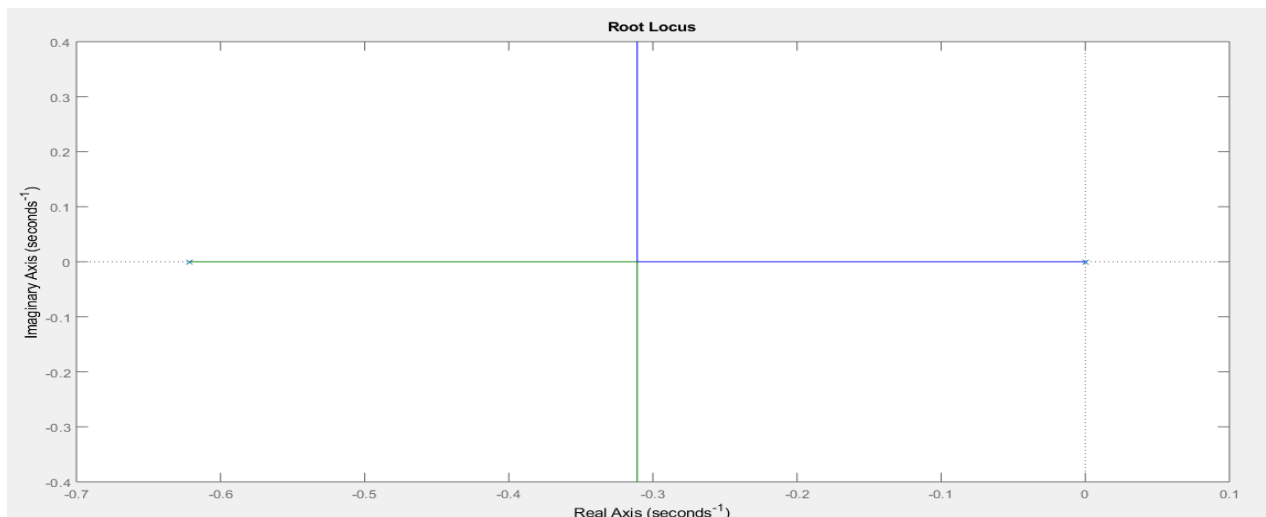


Figure 8: Root locus analysis of open loop poles

To gauge an estimation of how fast the system needs to be, the open loop settling time is calculated.

$$t_{2\%} = 4\tau = 4 \times 1.6082 = 6.43s$$

The system is much too slow and thus will not be able to track setpoints accurately, a controller needs to be designed to achieve a faster closed loop response.

## System Structure

The solution to the control problem at hand needs to be one that allows for the model to be controlled with zero steady state error in position and with a response faster than the closed loop uncontrolled plant. The system is stable, it's a SISO system and it contains non-linearities that can be compensated for. Therefore, control techniques learnt up until now can be used to design a controller to aid in allowing for the system to meet technical specifications.

The types of controller configurations that can be used are:

- Proportional controllers
- Integral controllers
- Derivative controllers
- State Feedback controllers

The components in the system which do not depend continuously on their inputs are the components which relate to the non-linearity elements in the system. The DAC is independent of its input when driven into saturation, it will produce the same output regardless of what input is fed into it when in saturation. The output potentiometer is responsible for the deadband, in order to start moving it needs to have enough energy to overcome static friction, and if the input is too small, will not move.

## State Feedback Controller Design

### Controller Selection

In designing the controller, a key consideration was having to decide between which controller type was best suited to the problem at hand. Essentially, a choice had to be made between designing a controller using the classical control method of design (P, PI, PID etc.) or the modern/optimal control design method relating to state feedback control. Bearing in mind the technical specifications, the steady state error, settling time and degree of overshoot are parameters on which the performance level of each controller was based. It also should be noted that classical control methods are best suited for linear SISO systems.

Even though the DC servomotor system is a SISO system, the soft non-linearities present in the system (which have been linearized and approximated as straight lines for the purpose of this project) together with the hard non-linearities such as the saturation and dead-band prevalent in the system suggest that the state feedback control technique would suit it better. This is especially considering that in the latter stages of this project, when model predictive control techniques are exploited and used in the controller design, state space control techniques will have to be used in any case.

In terms of controller performance, classical control methods and optimal control methods both 'solve' control problems, however, optimal control is an extension of classical control in that the controller design is based on optimizing a certain set of variables. In the DC servomotor system, a key requirement is to have the closed loop not slower than the uncontrolled plant. Knowing that, in the real world, the system is vulnerable to both input and output disturbances, the best way to ensure robustness of the controller is to obtain an optimal settling time whilst still tracking setpoints and maintaining stability. Classical control techniques carry out processes far away from optimum points to avoid instability, this leads to some of the technical specifications having to be relaxed. Optimal control allows for the process to be carried out close to the optimal point. Also, with state feedback control, steady state errors are relatively small and unnecessary disturbances and noise signals from

outside the system can be rejected with good recovery time whereas some classical control techniques do not reject input disturbances altogether or reject input disturbances with longer recovery periods.

Therefore, the state feedback controller design method was selected for design and implementation.

### Controller Design

The following block diagram depicts the desired state feedback model process, the outer loop is introduced into the system to achieve setpoint tracking.

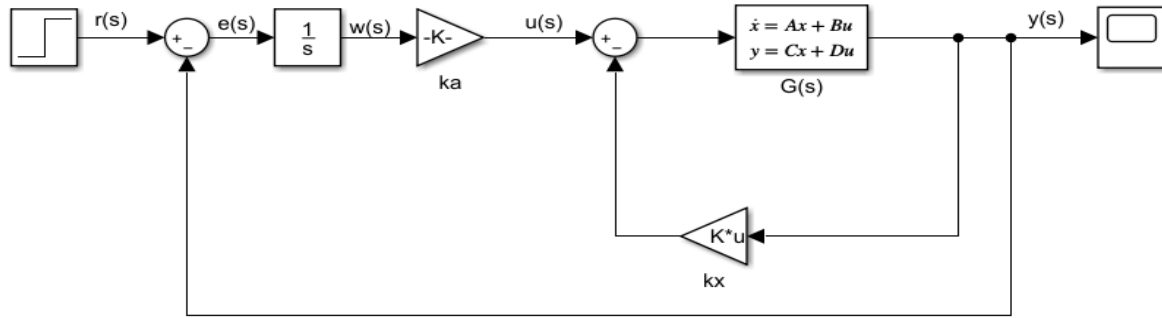


Figure 10: Block diagram of process model using state feedback control

Using direct programming, the process transfer function model was related to a state space model as seen below:

$$G(s) = \frac{y(s)}{u(s)} = \frac{83.515}{s(1 + 1.608s)} \times \frac{w(s)}{w(s)}$$

Defining states:

$$x(s) = \begin{bmatrix} sw(s) \\ w(s) \end{bmatrix}$$

Therefore:

$$sx(s) = \begin{bmatrix} 0 & 1 \\ 0 & -0.622 \end{bmatrix} x(s) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(s) = Ax(s) + Bu(s)$$

$$y(s) = [51.931 \ 0]x(s) = c^T x(s) + D$$

However, in order to calculate the gains associated with the closed loop system, the desired closed loop pole characteristic function  $\phi_c(s)$  needs to be equated with the determinant matrix given by  $|sI - F|$  in order to solve for  $k_T$ .  $k_T$  is a matrix which consists of three gains respectively, the controller gains  $k_1$  and  $k_2$  represented by ' $k*u$ ' in figure 1, as well as the integrator gain  $k_a$  represented by ' $-ka$ ' in figure 1.

To achieve a response faster than the uncontrolled plant, a pair of poles need to be added to the system and placed to the left of the dominant pole of the transfer function, which is at  $s = -0.622$ . In order to maintain a small overshoot, an underdamped system with zeta being around 0.7 is required. This can be achieved by ensuring that complex conjugate poles are used in pole selection, complex conjugate poles also help in keeping the gain associated with the integrator small so as to not allow it to have a big effect on the system. associated. The poles that we selected are at  $s = -0.8 + 0.8j$  and  $s = -0.8 - 0.8j$ . Another pole needs to be introduced into the system to compensate for the third gain associated with the setpoint tracking,  $k_a$ , in order to minimize its effect on the system, it will be placed further away from the dominant pole at  $s = -1.2$ .

The characteristic equation is then given by:

$$\phi_c(s) = (s + 0.8 + 0.8j)(s + 0.8 - 0.8j)(s + 1.2)$$

$$\phi_c(s) = s^3 + 2.8s^2 + 3.2s + 1.536$$

Looking at figure 9, the state space model for the process block is given by:

$$\frac{d}{dt}x = Ax + bu$$

$$y = c^T x$$

The state space model for the integrator is:

$$\frac{d}{dt}w = r - y = -c^T x + r$$

$$w = w$$

Combining these two systems gives:

$$\frac{d}{dt} \begin{bmatrix} x \\ w \end{bmatrix} = \begin{bmatrix} A & 0 \\ -c^T & 0 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} + \begin{bmatrix} b \\ 0 \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r$$

The input to the process is defined by:

$$u = -k^T x - k_a w = \begin{bmatrix} -k^T & -k_a \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix}$$

Therefore, the state space model for the entire system is:

$$\frac{d}{dt} \begin{bmatrix} x \\ w \end{bmatrix} = \begin{bmatrix} A - bk^T & -bk_a \\ -c^T & 0 \end{bmatrix} \begin{bmatrix} x \\ w \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r$$

$$F = \begin{bmatrix} A - bk^T & -bk_a \\ -c^T & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -k_1 & 0.622 - k_2 & -k_a \\ -51.931 & 0 & 0 \end{bmatrix}$$

Therefore:

$$|sI - F| = \begin{vmatrix} s & -1 & 0 \\ k_1 & s + 0.622 + k_2 & k_a \\ 51.931 & 0 & 0 \end{vmatrix} = s^3 + (k_2 + 0.622)s^2 + k_1s - 51.931k_a$$

And by equating the characteristic equation  $\phi_c(s)$  with the equation above we have:

$$\phi_c(s) = |sI - F|$$

$$s^3 + 2.8s^2 + 3.2s + 1.536 = s^3 + (k_2 + 0.622)s^2 + k_1s - 51.931k_a$$

$$\therefore k_1 = 3.200$$

$$k_2 = 2.178$$

$$k_a = -0.0296$$

### Controller Simulation

The following block diagram was developed in Simulink and used for controller analysis and simulation.

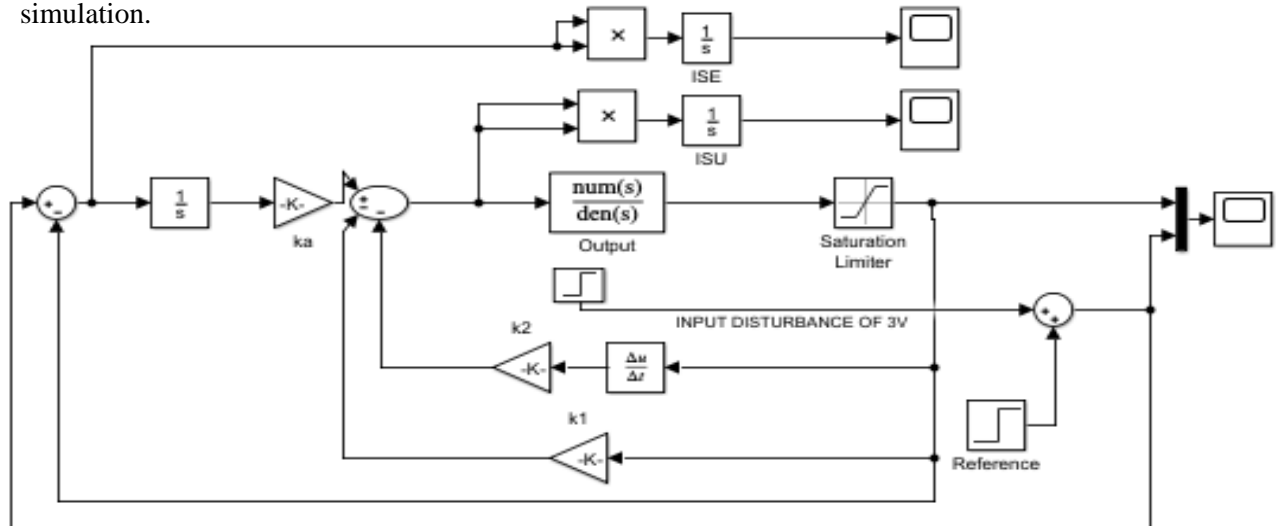


Figure 11: Block diagram used for controller simulation

As can be seen from figure 12 below, the response of the controller is in line with the technical specifications set out for it. The output has a settling time of 4.8s, there is no overshoot or ringing effects in the response and the setpoint is being tracked perfectly. Also, the output settles again after being influenced by the step disturbance at  $t=10$ s.

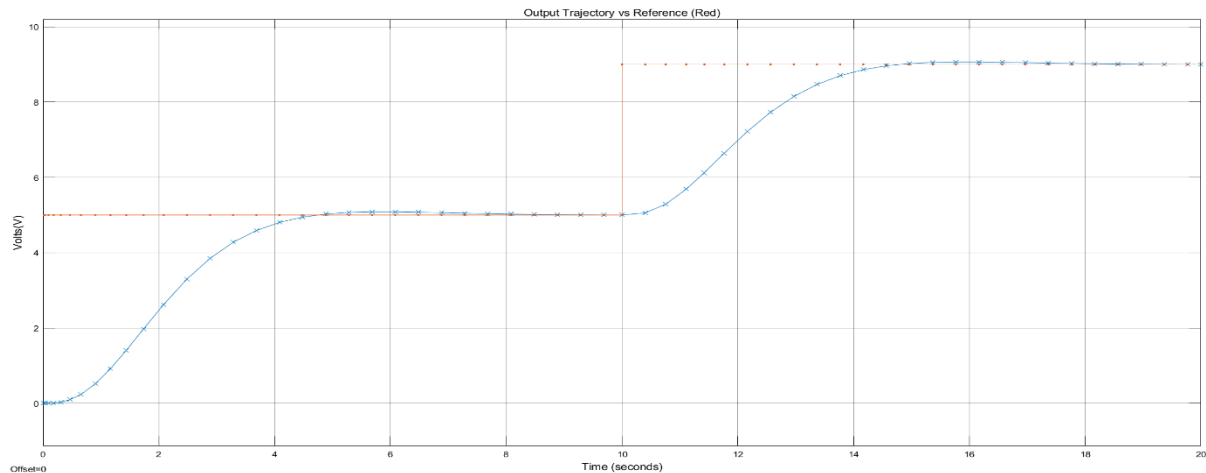


Figure 12: Simulated response of controller

Further system response analysis was done to ensure that the controller compensated for the effects of non-linearities, and to analyze their effects on the overall system response. The outcomes of this analysis can be found in Appendix A.

## State Feedback Controller Implementation

### Conversion Into the z-Domain

The controller is meant to be implemented digitally, therefore requiring the transfer function and associated gain values to be converted from the s-domain into the z-domain. A sampling time of  $T=0.1$ s was chosen. A snippet of the code used for this can be found in figure 13 below. The respective parameters relating to the pole placement and its associated s-domain gains were discretized using Euler discretization in Matlab and the Matlab 'place' function was used to find the z-domain gains. The corresponding z-domain gains that were used in the C code for implementation are:

$$k_1 = 2.9872$$

$$k_2 = 2.1057$$

$$k_a = -0.0257$$

The associated gains were then used in the following C code, which was designed to implement the controller and to cater for the non-linearities associated with the saturation and deadband.

```
// Controller
//Saturation Control
if (rt_SetPoint > 10)
{
    rt_SetPoint = 9.8;
}
else
{
    if (rt_SetPoint < -10)
    {
        rt_SetPoint = -9.8;
    }
}

double K_1 = 2.9872;
double K_2 = 2.1057;
double K_a = -0.2257;

double dxa_dt = rt_SetPoint - yt_PlantOutput;
double x_a = dxa_dt + x_a_prev;
x_a_prev = x_a;

double ut_Output = (-1 * K_a * x_a - ((yt_PlantOutput) * K_1 + K_2 * Velocity))/1.7;

//Deadband control
if (ut_Output < 0.3 && ut_Output > 0)
{
    ut_Output = 0.31;
}
else
{
    if (ut_Output < 0 && ut_Output > -0.9)
    {
        ut_Output = 0.91;
    }
}
```

Figure 13: C code used for controller implementation

As predicted, the resulting response was tracking the system, but the controller was ringing, and the controller did not settle after 2 seconds. The value of  $k_a$ , the integrator's gain which ought to have minimal effect on the system, was tweaked with through trial and error and was eventually increased to  $k_a = -0.2257$  whilst maintaining that it doesn't have a significant impact on the systems response, and the gain of the system was decreased by dividing  $ut\_output$  by 1.7. The gain needed to be decreased because it was noted that systems with low attenuation values and, therefore, high gains resulted in disturbances being amplified. This resulted in much more ringing than the same system would produce after altering the attenuation to a higher value. This implies that the higher the gain associated with the system, the more vulnerable it is to disturbances. The final controller that was designed and tested by Mr. de Maar resulted in the following response:

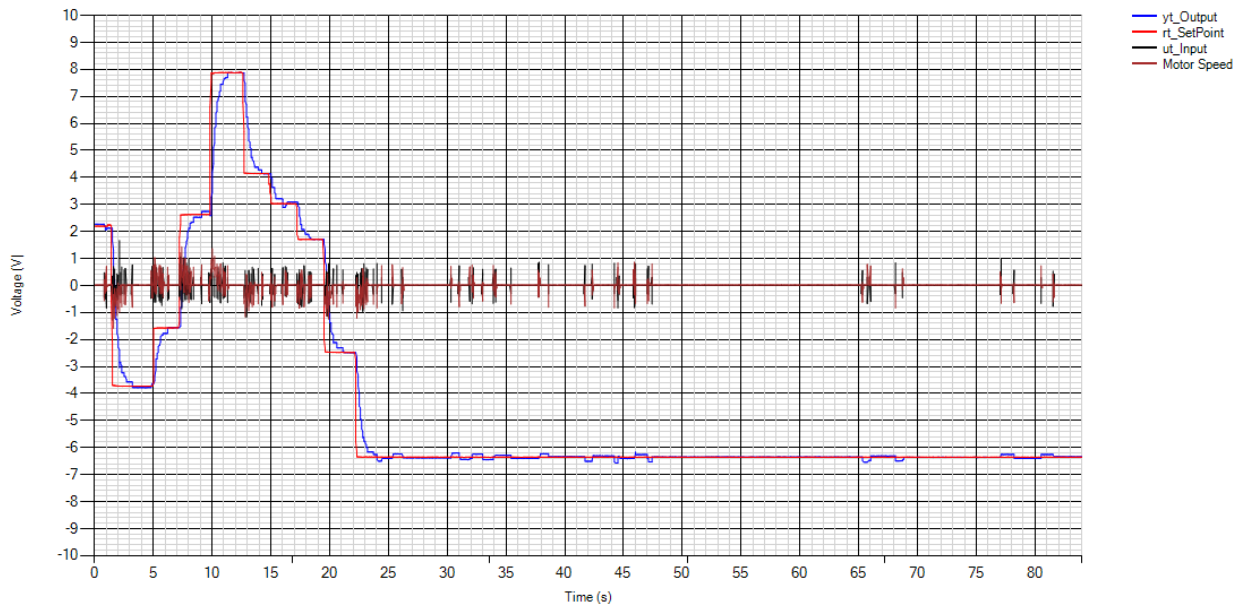


Figure 14: Response of implemented controller

## State Feedback Controller Experimental Results

The implemented response behaved almost exactly as the simulated response predicted. The implemented system settles faster than expected at around 2s. It rejects disturbances, carries very little overshoot and tracks the setpoints perfectly. However, the simulated response did not account for the ringing in the system which is prevalent in the implemented response, the reasons behind this ringing is stated in the previous section but could also be due to the fact that a system implemented in the real world will always have factors of influence that are beyond control in developing the simulated controller. This ringing, however, was considered to be negligible by Mr. de Maar and the technical specifications were fully achieved and the modelling and design methods used for this project resulted in the development of a robust and successful controller.

## MPC Controller Design

### Controller Design

The following block diagram illustrates the process behind designing the intended MPC controller. The introduction of the state observer is fundamental in allowing for present state variable manipulation in state feedback control, especially because the states aren't changing in sync. Designing an observer for all states, therefore, enforces zero steady state error and disturbance rejection.

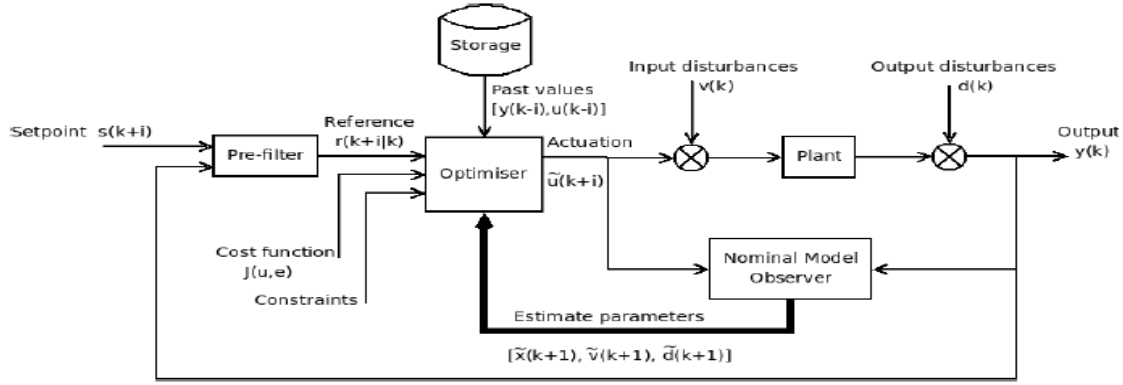


Figure 15: Block diagram used for MPC configuration

The entire procedure involved in designing the controller for implementation can best be represented by the flow chart below. The respective mathematical breakdown for some of the blocks in the flow chart are given below.

Using Euler's method of discretization, the discrete plant model was obtained to give  $A^*$ ,  $B^*$ ,  $c^*$  and  $D^*$ . Thereafter, the plant was augmented to cater for input disturbances as seen below:

$$\begin{bmatrix} x(k+1) \\ v(k+1) \end{bmatrix} = \begin{bmatrix} A^* & B^* \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x(k) \\ v(k) \end{bmatrix} + \begin{bmatrix} B^* \\ 0 \end{bmatrix} \begin{bmatrix} u(k) \\ \rho \end{bmatrix}$$

$$y = [c^{*T} \quad 0 \quad 0] \begin{bmatrix} x(k) \\ v(k) \end{bmatrix} + Du$$

Assume that the state space model parameters for the system above are represented by  $A'$ ,  $B'$ ,  $c'$  and  $D'$ . The observer poles are chosen to be twice as fast as or faster than the closed loop poles, or in this case positioned to the left of  $2 \times \frac{-1}{1.6082} = 1.25$ . Therefore, the poles are placed at  $s = -2.5$ ,  $s = -3.5$  and  $s = -4.5$ . Thereafter, after finding the respective z-domain pole locations, using a similar approach as the method used in the classical controller design, the desired closed loop pole characteristic function  $\phi_c(z)$  is equated with Ackermann's formula given by  $|zI - A' + Lc^T|$  in order to solve for the observer gain matrix  $L$ .

The next step involves choosing tuning parameters. Assuming a linear cost function, initially there was no need to place emphasis on either the error or control penalty, so  $\rho = \lambda = 1$ . The prediction horizon,  $n_y$ , was set to 20 with the control horizon,  $n_u$ , set to 19. Also,  $T_r = 0.1$ . Thereafter, the prediction modeling calculations follow as such:

$$\vec{y}_k = G\Delta\vec{u}_{k-1} + \vec{f}_k$$

$$\vec{f}_k = \begin{bmatrix} c^T A \\ \vdots \\ c^T A^{n_y} \end{bmatrix} x(k) + \begin{pmatrix} c^T B & \cdots & 0 \\ \vdots & \ddots & \vdots \\ c^T A^{n_y-1} B & \cdots & c^T B \end{pmatrix} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} u(k-1)$$

$$G\Delta\vec{u}_{k-1} = \begin{pmatrix} c^T B & \cdots & 0 \\ \vdots & \ddots & \vdots \\ c^T A^{n_y-1} B & \cdots & c^T B \end{pmatrix} \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix} \begin{bmatrix} u(k) \\ \vdots \\ u(k+n_y-1) \end{bmatrix}$$

After computing the above, the matrix 'G' can be found and used in order to calculate the system gain values using the control law. This gives:

$$K = (G^T Q G + R)^{-1} G^T Q \quad Q = I\lambda \quad R = I\rho$$

With both Q and R being matrices of size  $n_y$ .

The free response,  $\vec{f}_k$ , is calculated using the formula given above. A reference trajectory vector of size  $n_y$  is created with  $i$  incremented from 1 to  $n_y$  the reference trajectory formula for a setpoint vector,  $s(k+i)$ , is:



$$r(k + ilk) = s(k + i) - e^{-\frac{iT_s}{T_r}}(s(k) - y(k))$$

Then, the control action for each incremental is given by:

$$u(k) = u(k - 1) + \Delta u(k)$$

$$\Delta u(k) = K(\vec{r}(k + ilk) - \vec{f}_k)$$

Finally, the system model output is given by:

$$x(k + ilk) = Ax(k) + Bu(k) + L(y_{plant} - y(k))$$

$$y(k + ilk) = c^T x(k + ilk)$$

The last thing to do is to ensure that the plant stays between the saturation limits, and that the plant is forced to produce an output when in the deadband range. An explanation as to how this was compensated for through code is given in Appendix A.

### Controller Simulation

Using the above controller design methodology, an MPC controller was designed and simulated in Scilab using the code attached in Appendix B. The results follow:

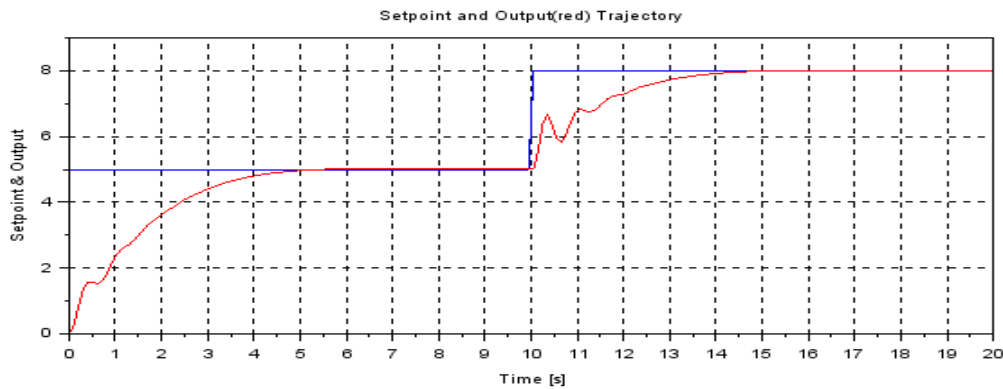


Figure 16: Graphs representing the simulated output response

From figure 16, the designed controller meets the technical specifications required of it. Its output trajectory manages to settle within 5s without any overshoot or ringing. There is no steady state error in tracking the fixed position setpoint, and when induced by a step input disturbance at  $t=10s$ , the output manages to settle again. The closed loop is also faster than the uncontrolled plant (6.43s). The response when the system is induced by external input and output disturbances is seen below:

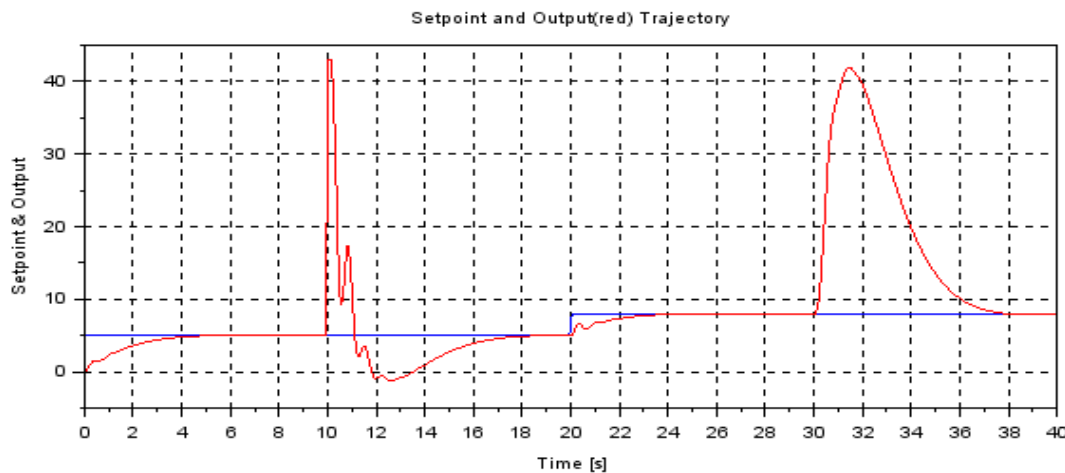


Figure 17: Response to external input and output disturbances

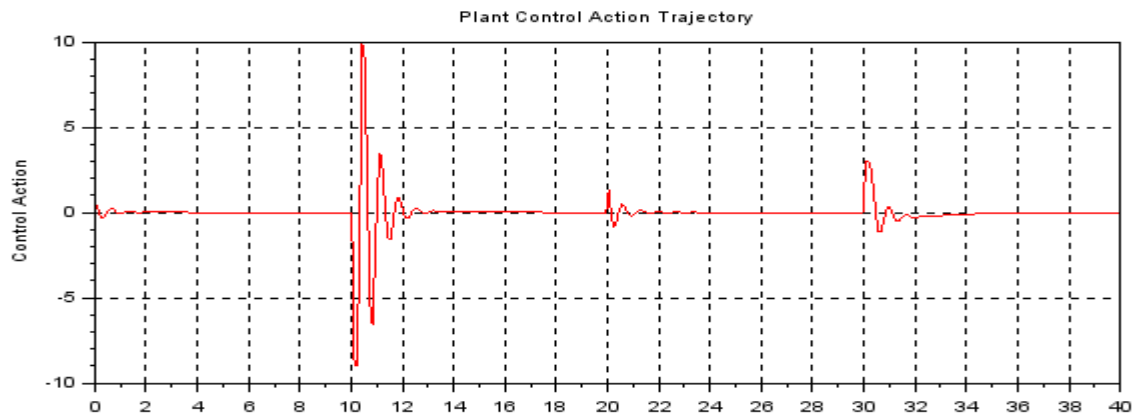


Figure 18: Control action response to external disturbances

Figure 17 and figure 18 show that the controller manages to compensate for the input disturbance at  $t=30s$  and allows for the output to settle relative quickly, and compensates for the output disturbance at  $t=10s$  by wrapping around without being forced into saturation.

## MPC Controller Implementation and Results

The Scilab code used to find a simulated response was then used as a basis to for controller design in C, and the syntax was changed accordingly. The corresponding C code for implementation is given in Appendix B. Once implemented in the control lab and tested by Mr De Maar, the following results were obtained.

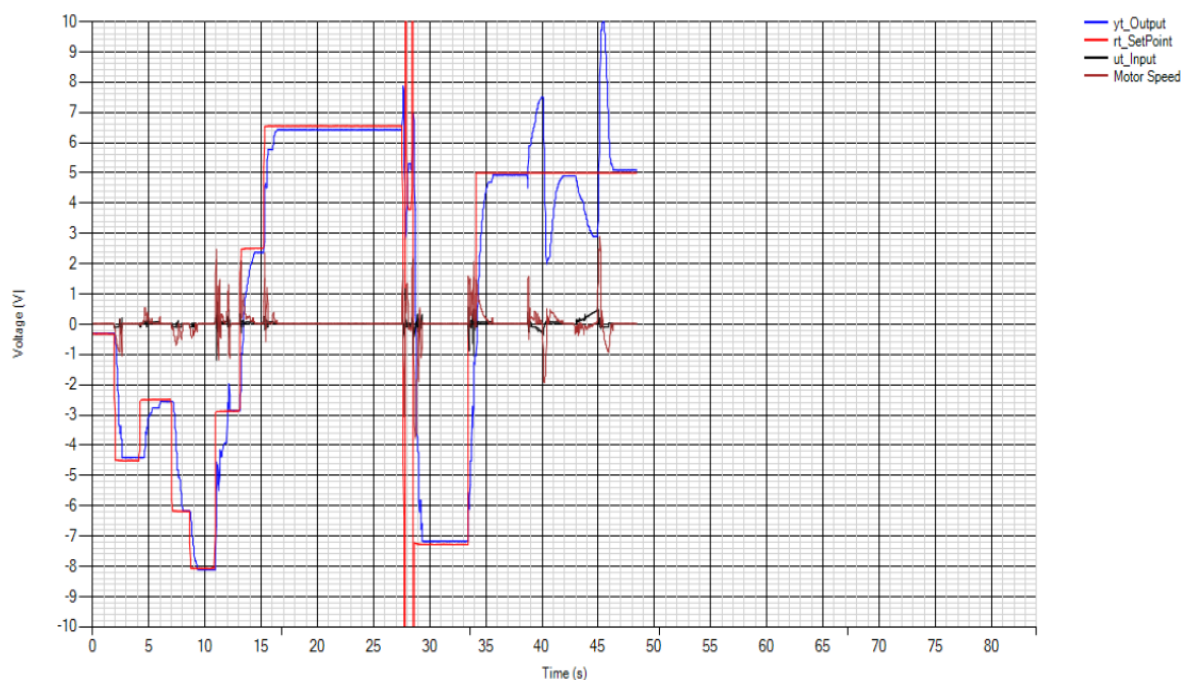


Figure 19: Graph representing the MPC controller implementation

## MPC Experimental Results

From figure 18 above, as expected from the Scilab simulation results, the designed MPC controller does indeed meet technical specifications. The output settles much faster when compared to the simulated response, however, settling within just over 1s. There is minimal control action required in dealing with all disturbances. When subjected to a significantly large output disturbance at  $t=27s$ , the output responds accordingly by wrapping around and settling again in under 3s. The same response occurs when the system is subjected to another disturbance at  $t=45s$ , The output manages to avoid saturation before settling again. There is, however, a small steady state error present in the output when trying to track the setpoint.

## Controller Comparison and Conclusion

The project was indeed a success, and both the state feedback and MPC controllers matched up almost perfectly with the technical specifications. The following table compares the controller performances against each other for both controller simulation and implementation. Graphs depicting the ISU and ISE results for each controller are given in Appendix B.

### Simulated Response

Performance Criterion	MPC Controller	State Feedback Controller
1. Zero Steady State Error	Yes	Yes
2. Disturbance Rejection	Yes	Yes
3. < 20% Overshoot in Disturbance Rejection	Yes	Yes
4. Faster Than Uncontrolled Plant	Yes	Yes
5. Settling Time	4.8s	4.8s
6. Transient ISE	225	39
7. Steady State ISE	0	0
8. 3v Step Disturbance ISE	52	25
9. Transient ISU	0.79	0.0024
10. Steady State ISU	0	0
11. 3v Step Disturbance ISU	3.47	0.0015

Figure 20: Simulated MPC vs Simulated State Feedback Controller

According to figure 20, the MPC controller and State Feedback controller perform identically in terms of how well they achieve technical specifications and requirements. However, the State Feedback controller far surpasses the MPC controller in terms of the ISU and ISE parameters.

### Implemented Response

Performance Criterion	MPC Controller	State Feedback Controller
1. Zero Steady State Error	<0.2	Yes
2. Disturbance Rejection	Yes	Yes
3. < 20% Overshoot in Disturbance Rejection	Yes	Yes
4. Faster Than Uncontrolled Plant	Yes	Yes
5. Settling Time	1.2s	2s

Figure 21: Implemented MPC vs Implemented State Feedback Controller

From figure 21, the only real parameter that sets the controllers apart is the settling time. The MPC controller, when implemented, settled 67% faster than the state feedback controller. The MPC controller did, however, carry a small amount of steady state error.

From the results above, at least based on the scope of this project, it can be concluded that MPC controllers and State Feedback controllers are both suitable in designing and implementing a digital control system for position control of a DC Motor. In simulation, the State Feedback controller excels in maintaining lower far lower ISU and ISE values compared to the MPC controller. In implementation, the MPC controller does do better with regards to the speed of the controller, but the State Feedback controller does manage to track setpoints somewhat better.

In hindsight, perhaps designing the MPC controller using CAD tools would allow for the MPC controller to be a lot more robust, especially in terms of its ISU and ISE parameters. The optimization

techniques for MPC control allow for these parameters to be optimized simultaneously, together with all the other constraints and parameters of the plant. The MIMO characteristics of model predictive control allow for it to handle multivariable optimization in a way that no other classical controller design method can.

## References

Braae, M., 2011. *Control Engineering-1*. 3rd ed. s.l.:s.n.

Mohan, N., 2002. *Power Electronics: Converters, Applications and Design*. 3rd ed. s.l.:s.n.

Tseou, M., 2019. *Augmented Constrained MPC*, s.l.: s.n.

## Appendix A – Classical Control Design

The Simulink block diagram in figure 11 is augmented to analyze the effects of the saturation non-linearity constraint of the controller, after compensating for it.

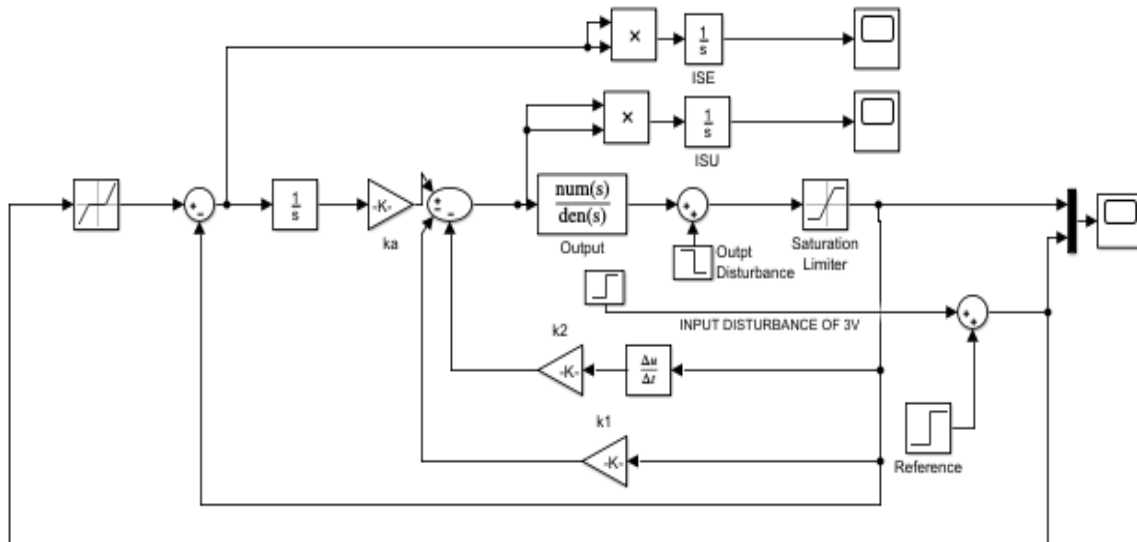


Figure 22: Block diagram used for non-linearity analysis

The saturation block is meant to provide an output between -9.9V and 9.9V if the input provided to the system consists of voltage values above or below 10V. In the simulation, a 0.1V leeway was used to compensate for the slight overshoot. However, when coding the controller's implementation, a 0.2V leeway was assumed for the controller to compensate for any practical concerns or disturbances that might affect the system, the system designed and tested in real life would therefore saturate at 9.8V instead.

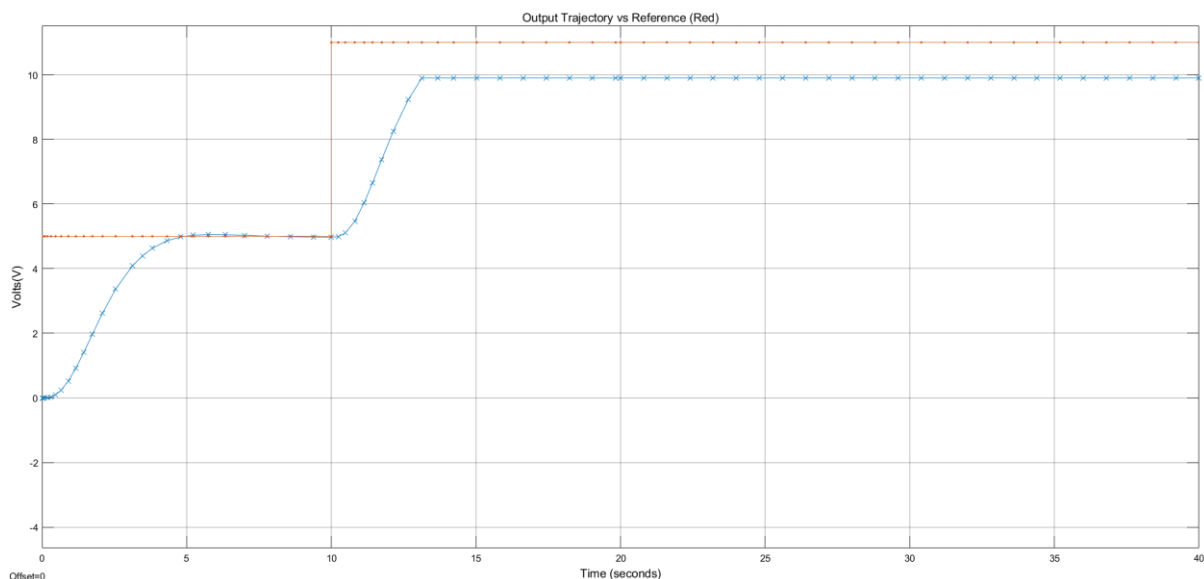


Figure 23: Controller response to inputs higher than 10V

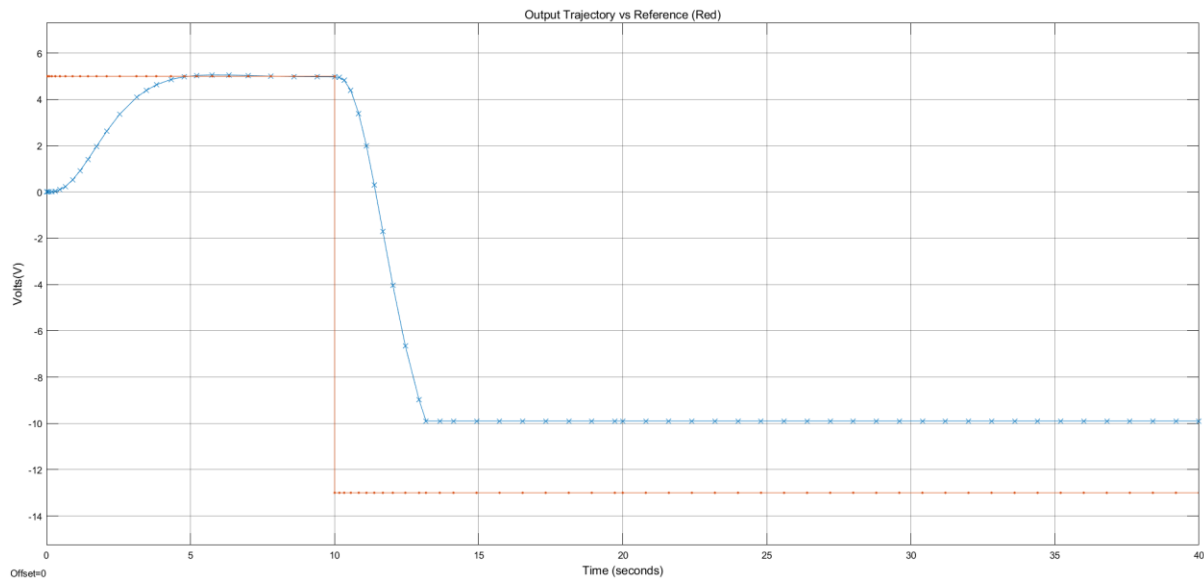


Figure 24: Controller response to inputs lower than 10V

The effects of any input and output disturbances on the controller after compensating for them are now analyzed. A step input value of 4V was used as a disturbance and was fed into the system at  $t = 5s$  for the input disturbance, and a step input value of -5V at  $t=15s$  was used as an output disturbance. The response is given below:

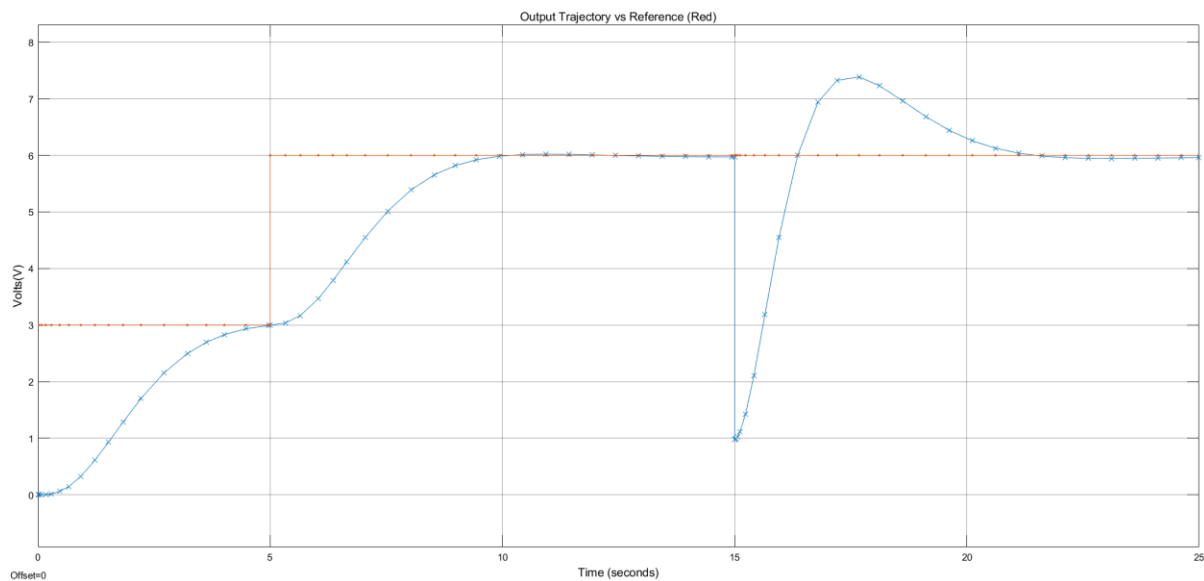


Figure 25: Effects of disturbances on system response

The system manages to track the setpoint very rapidly and provides an adequate disturbance rejection which satisfies the disturbance rejection specifications.

```

syms k1 k2 ka s z

%Input values
tau = 1.6082;
A = 83.515;

%continuous state space parameters
cont_A = [0,1;0,-1/tau];
cont_B = [0;1];
cont_C = [A/tau, 0];

%IMC
A_twiddle = [cont_A;-1.*cont_C];
A_twiddle = [A_twiddle zeros(size(A_twiddle,1),1),1 ]';

b_twiddle = [cont_B;0];
k_transpose = [k1, k2, ka];
c_twiddle = [cont_C, 0];
%Discretizing using Euler
T = 0.1;
F = (eye(3)+T.*A_twiddle);
G = T.*b_twiddle;

%Desired poles
s1 = -0.8 + 0.8j;
s2 = -0.8 - 0.8j;
s3 = -1.2;

z1 = exp(s1*T);
z2 = exp(s2*T);
z3 = exp(s3*T);

%k = place (A_twiddle, b_twiddle, [s1 s2 s3])
k = place (F, G, [z1 z2 z3])

```

Figure 26: Matlab code for controller design in z-domain

## Appendix B – MPC Controller Design

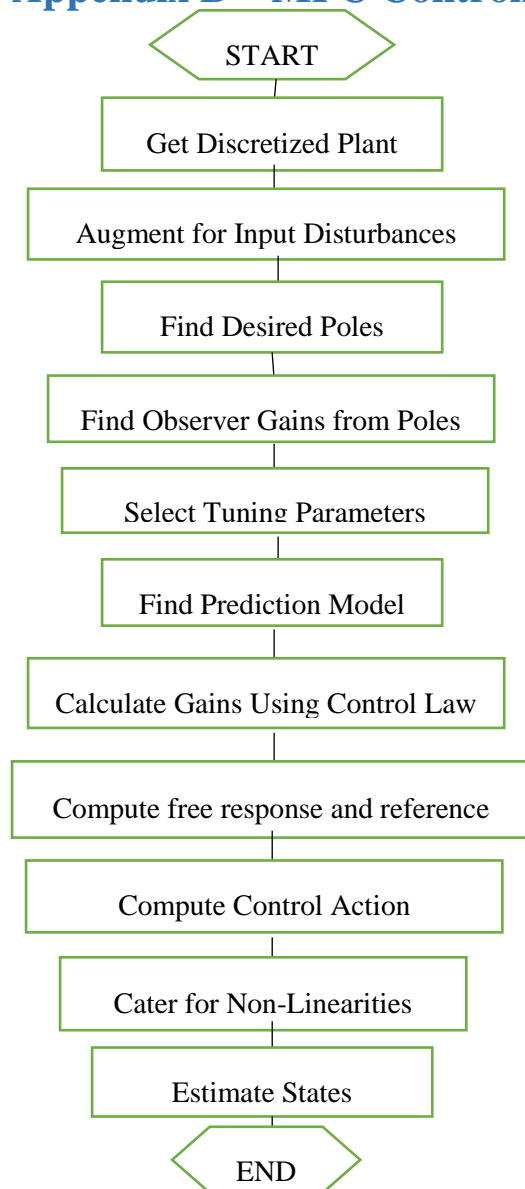


Figure 27: Flow chart representing MPC design methodology



The following code represents the Scilab code that was used to obtain the simulated MPC controller response and is based off the example we've been provided with by Dr Tseou as part of our course material.

```
s = %s;
//-----
// Real System
//-----
Ts = 0.1; // Sampling period [s]
plant_tf = syslin('c', (83.515)/((s^2)+1.6082*s)); // Laplace Transfer Function
plant_ss = dscr(plant_tf,Ts); // Discrete state space

//-----
// System's Nominal model augmented for disturbances
//-----

model_tf = syslin('c', (83.515)/((s^2)+1.6082*s));
model_ss = dscr(model_tf,Ts);

dummy0 = size(model_ss.a);
dummy1 = size(model_ss.b);
dummy2 = zeros(dummy1(1),dummy1(2));
dummy3 = zeros(2,dummy0(2));

tau = 1.6082;
ai = 83.515;
T = 0.1;
A = [1 T 0; 0 1-T/tau T; 0 0 1];
B = [0;T;0];
C = [ai/tau 0 0];
D = model_ss.d;

//-----
// Tuning parameters
//-----
ny = 20; // Prediction horizon
nu = 19; // Control horizon
P = [1:20]'; // Coincidence points set
Tr = 1; // Reference trajectory time constant
rho = 1; // Error penalty
lambda = 1; // Control penalty
L = [0.015727; 0.038842; 0.045582]; // Observer gains

sizeP = size(P);

//-----
// Prediction Model
//-----
Py1 = [];
Huy = [];
Hduy = [];
temp1 = 0;

for n = 1:ny,
    Py1 = [Py1;C*A^n];
    temp1 = [temp1+C*A^(n-1)*B];
    Huy = [Huy;temp1];
end

Hduy = Huy;
temp = Huy;

for i = 1:nu,
    temp = [0;temp(1:ny-1)];
    Hduy = [Hduy,temp];
end
```

```

G = Hduy;

//-----
// The Control Law
//-----

Q = rho*eye(ny,ny); // Error penalty matrix
TMP = G'*Q*G;
sizeTMP = size(TMP);
R = zeros(sizeTMP(1),sizeTMP(2)); // Control penalty matrix
for i = 1:ny,
    R(i,i) = lambda;
end

K = inv(G'*Q*G + R)*G'*Q; // Control law

//-----
// Initial Values
//-----
endSim = 200;
tv = Ts* linspace(0,endSim, endSim);
u0 = 0;
x0 = zeros(dummy0(1)+1,1); //Observer States (plant states + disturbances)
xP = zeros(dummy0(1),1); // Plant states
ref = [];
r = [];
u = [];
up = [];
ap=[];
y = [];
e = [];
ise = [];
isu=[];
temps = 0;
temps2 = 0;
sp = 5*ones(1,endSim+ny+1);
e0 = sp(1);
uSatMin = -10;
uSatMax = 10;

rand('uniform');
//-----
// Control Cycles
//-----
for k = 1:endSim,
    if k < 2,
        pastU = u0;
        xNow = x0;
        eNow = e0;
    end
    e(k) = eNow;
    errr = temps;
    ise(k) = errr + eNow.^2;
    temps = ise(k);

    // step disturbances
    if k > 1*endSim/2,
        sp(k) = sp(k)+3;
    else,
        sp(k)= sp(k);
    end
    // Compute the free response and reference
    r = (sp(k*ones(sizeP(1),1)))' - ((exp(-P*(Ts/Tr))) * eNow); // Reference
    f = Py1(P,:)*xNow + Huy(P,:)*pastU; // Free response
    ref(k) = r(1);

```

```

// Compute the control action and update storage
delU = K(:,P)*(r - f);
u(k) = pastU + delU(1);
pastU = u(k);
tempU = u(k);

// Drive the plant with input

// Plant saturation
up(k) = u(k);

urrr = temps2;
isu(k) = urrr + u(k).^2;
temps2 = isu(k);

if u(k) > uSatMax,
    up(k) = 9;
end
if u(k) < uSatMin,
    up(k) = -9;
end

// Include input disturbances
if k > 2*endSim/3,
    up(k) = up(k)+0;
else,
    up(k) = up(k);
end

// Include output disturbances
if k > endSim/3,
    y(k) = plant_ss.c*xP + plant_ss.d*up(k)+ plant_ss.d*up(k)+0;
else,
    y(k) = plant_ss.c*xP + plant_ss.d*up(k);
end

xP = plant_ss.a*xP + plant_ss.b*up(k);

// Estimate the states of the system and disturbances
yP = C*xNow + D*tempU;
xNow = A*xNow + B*tempU+L*(y(k) - yP);

// Compute the error and add noise
eNow = sp(k) - y(k);

end
for k = 1:endSim,
    ap(k) = sp(k);
end

//-----
// Present Results
//-----
clf;
subplot(2,2,1)
plot(tv,ap')
plot(tv,y,'r')
title('Setpoint and Output(red) Trajectory')
xlabel('Time [s]')
ylabel('Setpoint & Output')
xgrid()
subplot(2,2,2)
plot(tv,ref)
title('Reference Trajectory')
xlabel('Time [s]')

```

```

ylabel('Reference')
xgrid()
subplot(2,2,3)
//plot(tv,u)
plot(tv,isu,'r')
title('ISU')
xlabel('Time [s]')
ylabel('Control Action')
xgrid()
subplot(2,2,4)
plot(tv,ise)
title('ISE')
xlabel('Time [s]')
ylabel('Error')
xgrid()

```

(Tseou, 2019)

The following code represents the C code that was used to obtain the implemented MPC controller response.

```

// Global control variables

double yt_PlantOutput;      // Output of plant for position @ ADC INPUT 0 (Temperature output from sensor)
double rt_SetPoint = 0;     // Setpoint @ ADC INPUT 1 //Main global variable to have variable with regards to
time
double rt_Gain = 0;
double ut_PlantInput;      // Output of Controller to heater input @ ADC INPUT 2

double Gain = 1;           // Gain Default to one

double Ts = 0.1;
double Tr = 1;

//Plant State Space
Matrix<double> A_ss = DenseMatrix.OfArray(new double[,] {
{1,0.1,0},
{0,0.93782,0.1},
{0,0,1}});

Vector<double> B_ss = Vector<double>.Build.Dense(new[] { 0, 0.1, 0 });

static Vector<double> C_ss = Vector<double>.Build.Dense(new[] { 51.931, 0, 0 });
Matrix<double> C_T = C_ss.ToRowMatrix();

//Prediction Model
Vector<double> Huy = Vector<double>.Build.Dense(new[] { 0, 0.519307300087054, 1.52563068647637,
2.9886868557043, 4.88007556011269, 7.17316251695184, 9.84296959835019, 12.8660718494166,
16.2205009098931, 19.885654441177, 23.8422111852898, 28.0720513055926, 32.5581816808201,
37.2846658444303, 42.236558280419, 47.3998428047058, 52.7613747780483, 58.3088269122326,
64.030638446107, 69.9159674819168 });
Vector<double> r = Vector<double>.Build.Dense(20);
Vector<double> f = Vector<double>.Build.Dense(20);

Vector<double> Xp = Vector<double>.Build.Dense(2);

double pastU = 0;
double tempU = 0;
Vector<double> yP = Vector<double>.Build.Dense(new[] { 0.0 });

Matrix<double> Py = DenseMatrix.OfArray(new double[,] {
{51.9307300087054,5.19307300087054,0},
{51.9307300087054,10.0632338638931,0.519307300087054},
{51.9307300087054,14.6305616922793,1.52563068647637},
{51.9307300087054,18.9138870440839,2.9886868557043},
{51.9307300087054,22.9308695683916,4.88007556011269},

```

```
{51.9307300087054,26.6980708139834,7.17316251695185},
{51.9307300087054,30.231022510664,9.84296959835019},
{51.9307300087054,33.5442906047652,12.8660718494166},
{51.9307300087054,36.6515353128385,16.2205009098931},
{51.9307300087054,39.5655674411286,19.885654441177},
{51.9307300087054,42.2984012030283,23.8422111852898},
{51.9307300087054,44.8613037522742,28.0720513055927},
{51.9307300087054,47.2648416361024,32.5581816808201},
{51.9307300087054,49.5189243598866,37.2846658444303},
{51.9307300087054,51.6328452428684,42.236558280419},
{51.9307300087054,53.615319733425,47.3998428047058},
{51.9307300087054,55.4745213418428,52.7613747780483},
{51.9307300087054,57.2181153387435,58.3088269122326},
{51.9307300087054,58.8532903580979,64.030638446107},
{51.9307300087054,60.386788035122,69.9159674819168}};
```

```
Vector<double> X_now = Vector<double>.Build.Dense(new[] { 0.0, 0.0, 0.0 });
Vector<double> U_del = Vector<double>.Build.Dense(20);
```

```
//Control Law
Matrix<double> K = DenseMatrix.OfArray(new double[,] {
{0,0.106162642088155,0.153632497010514,0.129577906851966,0.0736482817642027,0.0228264836005438,-
0.00646823257768831,-0.0151836852027801,-0.0119458289914144,-0.00526148518536078,-
8.68162329615046e-
05,0.002183684016906,0.00222988655192209,0.00130231083480946,0.000348703505682231,-
0.00021940281826042,-0.000387847692150368,-0.000302365784949439,-9.48607423882919e-
05,0.000178848105788276},
{0,-0.158254095956505,-0.122853543073961,-
0.0395259750717667,0.0197923890809513,0.0396213276988735,0.0324678316919529,0.0161642269625017,0.
0026218009240119,-0.00410350076106437,-0.00512956292074651,-0.00333399303006358,-
0.00112727525712988,0.000300939171240714,0.000781848604807876,0.00064756605433158,0.000299787856
014218,3.57838201338881e-06,-0.000133193828183664,-9.48607423865155e-05},
{0,-0.0164812529386289,-0.182104864629471,-0.142970009947899,-
0.0509596338757502,0.0162487184886652,0.0406258567860161,0.0348258204491259,0.0180197547672585,0.
00343906798787227,-0.00409135286598461,-0.00547280668170222,-0.00368710952702374,-
0.00133601714743388,0.000247153749156284,0.000830864952011702,0.000739052291397221,0.00037818311
4660534,3.57838199604998e-06,-0.000302365784950354},
{0,0.0352019256919606,0.034460820973814,-0.139139122164268,-0.118549698432694,-
0.0433906087144231,0.0141051179325051,0.0355937033833592,0.0308679320423983,0.0162765479302914,0.
00340604196409412,-0.00338078174397994,-0.00475552169610083,-0.00327620625718141,-
0.00121927939203983,0.00022207198892779,0.000801947697154723,0.000739052291392157,0.000299787856
018696,-0.000387847692149279},
{0,0.0350008300831973,0.0858529467178062,0.0771810206266416,-0.114858414177139,-
0.111023880994975,-
0.0455216269405265,0.00910244016843316,0.0316593385773819,0.029135093649915,0.0162424879254409,0.
00410865857314905,-0.0026739718494356,-0.00435300052257779,-0.00315981438141508,-
0.00123046906948255,0.00022207198893138,0.000830864952019962,0.000647566054336579,-
0.00021940281826005},
{0,0.0167368664854756,0.059221365268778,0.106281061617824,0.0887916485501543,-0.111259670146345,-
0.11204277056512,-
0.0479135479943301,0.00722144211644265,0.0308308811407581,0.0291183308445476,0.0165769525801123,
0.00444416236552446,-0.00248384092120432,-0.00429721876529408,-0.003159814381406,-
0.00121927939204043,0.00024715374913807,0.000781848604820112,0.000348703505679282},
{0,0.000390763957165309,0.0173025161329862,0.0596986976342014,0.106552543618002,0.088875543132380
3,-0.111284824857445,-0.112101572843335,-
0.0479611714401086,0.00720043565425044,0.0308354503025635,0.0291419437325601,0.0166106606329422,
0.00447308572078089,-0.00248384092122422,-0.00435300052256391,-0.00327620625719318,-
0.00133601714745797,0.000300939171266279,0.00130231083480936},
{0,-0.0073463089293813,-
0.0102398020279127,0.00833732744962735,0.0546038206433649,0.104972514561221,0.0893181074699387,-
0.110245018054588,-0.111288581514604,-
0.0476032087014259,0.00722470121359572,0.030742511320745,0.0290828751020033,0.0166106606329464,0.
00444416236552045,-0.00267397184943849,-0.00475552169611192,-0.00368710952703785,-
0.00112727525712586,0.00222988655191614},
{0,-0.0077635487111976,-0.0185802590207596,-
0.019713242501818,0.0029540546153884,0.052933745911243,0.105436918763627,0.0904098373984805,-
```

```
0.109394795018882,-0.110914302533425,-  
0.0475655976126567,0.00716461098078896,0.0307425113207108,0.029141943732552,0.016576952580152,0.0  
0410865857315147,-0.00338078174395696,-0.00547280668171838,-  
0.00333399303008264,0.00218368401692004},  
{0,-0.004741855297808,-0.0146246981004851,-0.0243655909927154,-  
0.0230003265222357,0.00193370369904612,0.0532142240514061,0.106096863555975,0.0909205319035856,-  
0.109170055711342,-0.110879810228981,-  
0.0475655976126722,0.00722470121357669,0.0308354503025251,0.0291183308445885,0.0162424879254994,  
0.00340604196403481,-0.00409135286592693,-0.00512956292083722,-8.68162329172861e-05},  
{0,-0.00148836742516301,-0.00689577485745424,-0.0164414352537545,-0.0253982159911917,-  
0.0233203155836913,0.00202472153412154,0.0534278201432254,0.106265249913362,0.0909947045986632,-  
0.109170055711341,-0.1109143025334,-  
0.0476032087014376,0.00720043565425631,0.0308308811407603,0.0291350936499439,0.0162765479301813,  
0.00343906798802378,-0.00410350076117005,-0.00526148518533276},  
{0,0.0004992859421079,-0.00076821979356883,-0.00629212094313887,-0.0161010563228072,-  
0.0252889725702092,-  
0.023330420282154,0.00199728219349633,0.0534267603574554,0.106265249913408,0.0909205319035885,-  
0.109394795018882,-0.111288581514635,-  
0.0479611714401413,0.00722144211647674,0.0316593385773071,0.0308679320424212,0.0180197547673444,  
0.00262180092399422,-0.0119458289914425},  
{0,0.00111220969314972,0.00210338490326452,0.000576233599117664,-0.00553414919956245,-  
0.0158576246685727,-0.0253106117187381,-  
0.0233896610109365,0.00199728219349007,0.053427820143243,0.106096863555993,0.0904098373985056,-  
0.110245018054567,-0.112101572843377,-  
0.047913547994367,0.0091024401683312,0.0355937033835326,0.0348258204489877,0.0161642269626929,-  
0.0151836852028815},  
{0,0.000887195822299949,0.00238921044250637,0.00316965392968554,0.00117441949872906,-  
0.00533793576937158,-0.0158530522949715,-0.0253106117187344,-  
0.0233304202821846,0.00202472153409134,0.0532142240514091,0.1054369187637,0.0893181074700122,-  
0.111284824857455,-0.112042770565232,-  
0.0455216269405845,0.0141051179325826,0.0406258567858632,0.0324678316922675,-  
0.00646823257784401},  
{0,0.000409913937169148,0.00147746115658466,0.00288246056942406,0.00344665868355755,0.00126488010  
127965,-0.00533793576935428,-0.0158576246685567,-0.025288972570248,-  
0.0233203155836951,0.00193370369898707,0.0529337459112736,0.104972514561354,0.0888755431323307,-  
0.11125967014641,-0.111023880994927,-  
0.0433906087144923,0.0162487184886686,0.0396213276989292,0.022826483600523},  
{0,3.81314065164415e-  
05,0.00047546604448054,0.00154896903616936,0.00293490510797764,0.00344665868356199,0.00117441949  
873772,-0.00553414919953833,-0.0161010563227947,-0.0253982159912307,-  
0.0230003265222993,0.00295405461532345,0.0546038206434285,0.106552543618154,0.0887916485500991,-  
0.114858414177081,-0.118549698432759,-0.0509596338757635,0.019792389080812,0.0736482817643225},  
{0,-0.000120883192149768,-  
0.000103343969119401,0.000408472761853594,0.00154896903615078,0.00288246056942453,0.003169653929  
68318,0.000576233599106039,-0.00629212094302508,-0.0164414352537936,-0.0243655909927847,-  
0.0197132425019514,0.00833732744963711,0.0596986976343505,0.106281061617966,0.0771810206265447,-  
0.139139122164109,-0.142970009948096,-0.0395259750720069,0.129577906852191},  
{0,-0.000116114143112591,-0.000230047147681456,-  
0.000103343969114263,0.000475466044470489,0.00147746115658449,0.00238921044250011,0.002103384903  
23284,-0.000768219793432874,-0.00689577485749274,-0.0146246981005128,-0.0185802590209127,-  
0.0102398020279723,0.0173025161331189,0.0592213652689855,0.0858529467176995,0.0344608209740025,-  
0.182104864629608,-0.12285354307435,0.153632497010804},  
{0,-4.82317700345815e-05,-0.000116114143112381,-0.000120883192151864,3.81314065159375e-  
05,0.000409913937170355,0.0008871958222991,0.00111220969310627,0.000499285942195433,-  
0.0014883674251655,-0.00474185529783971,-0.00776354871125436,-  
0.007346308929425,0.000390763957242155,0.0167368664855684,0.0350008300830666,0.0352019256922164,-  
0.0164812529388946,-0.158254095956517,0.106162642088229},  
{0,0,0,0,0,
```

```
Vector<double> L = Vector<double>.Build.Dense(new[] { 0.015727, 0.038842, 0.045582 });
```

```
private void FormLoad(object sender, EventArgs e) // Connect to ADC/DAC
{
    //try
    //{
```

```

// //string DAQName;
// //string SerialNumber;
// //DAQ.ReadDAQ(out SerialNumber, out DAQName);
// //textBoxDAQName.Text = DAQName;
// //textBoxDAQSerial.Text = SerialNumber;

//}
//catch
//{
//    MessageBox.Show("uDAQ not connected", "Cannot read uDAQ",
//        MessageBoxButtons.OK, MessageBoxIcon.Error);
//}

deviceManager = DeviceMgr.Get();

if (device != null)
{ device.Dispose(); }

try
{
    device = deviceManager.GetDevice("DT9812(05)");
    output = device.AnalogOutputSubsystem(0);
    input = device.AnalogInputSubsystem(0);
    textBoxDAQName.Text = device.ToString();
    textBoxDAQSerial.Text = "Data Translation";
}
catch
{
    MessageBox.Show("DT9812 not connected", "Cannot read DT9812",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    textBoxDAQName.Text = "DAQ Fault";
    textBoxDAQSerial.Text = "Data Translation";
}

output.DataFlow = DataFlow.SingleValue;
input.DataFlow = DataFlow.SingleValue;
input.Config();
output.Config();
}

// Interrupt for timer
private void Timer_Tick(object sender, EventArgs e)
{
    string WriteData = "";
    double Increment = Convert.ToDouble(timer1.Interval) / 1000;
    XValue = XValue + Increment;
    //double XValue = count * timer1.Interval;

    // Timer recording
    double TimeElapsed = XValue;
    textBoxTimeElapsed.Clear();
    textBoxTimeElapsed.AppendText(TimeElapsed.ToString("0.00"));
    WriteData = TimeElapsed.ToString("0.00"); // Record into text string for data logging

    yt_PlantOutput = input.GetSingleValueAsVolts(0, 1); // Read and Display yt @ ADC INPUT 0
    textBoxPlantOutput.Text = yt_PlantOutput.ToString("0.000"); // Display yt
    WriteData += "," + yt_PlantOutput.ToString("0.000"); // Record into text string for data logging
    chart1.Series["yt_Output"].Points.AddXY(XValue, yt_PlantOutput);

    string InputSignal = comboBoxInputSignal.Text;
    if (InputSignal == "EXTERNAL(DAQ-1)")
    {
        rt_SetPoint = input.GetSingleValueAsVolts(1, 1); // Read and Setpoint @ ADC INPUT 1
    }
}

```

```

    }
    else
    {
        try
        {
            rt_SetPoint = Convert.ToDouble(rt_Gain) + TimeElapsed * Convert.ToDouble(textBoxRamp.Text);
        }
        catch
        {}
    }

    textBoxSetPoint.Text = rt_SetPoint.ToString("0.000");           // Display Setpoint
    WriteData += "," + rt_SetPoint.ToString("0.000");
    chart1.Series["rt_SetPoint"].Points.AddXY(XValue, rt_SetPoint);

    double Velocity = input.GetSingleValueAsVolts(2, 1);           // Read and Setpoint @ ADC INPUT 2
    textBoxMotorVelocity.Text = Velocity.ToString("0.000");
    WriteData += "," + Velocity.ToString("0.000");
    chart1.Series["Motor Speed"].Points.AddXY(XValue, Velocity);

    // Controller

    if (rt_SetPoint > 10)
    {
        rt_SetPoint = 9;
    }
    else
    {
        if (rt_SetPoint < -10)
        {
            rt_SetPoint = -9;
        }
    }

    //-----
    double eNow = rt_SetPoint - yt_PlantOutput;

    // Compute the free response and reference
    for (int i = 0; i < 20; i++)
    {
        r[i] = rt_SetPoint - Math.Exp(-(Ts * i) / Tr) * eNow;
    }

    Console.WriteLine("r: " + r.ToString());

    f = Py * X_now + Huy * pastU;

    Console.WriteLine("f: " + f.ToString());

    //Compute the control action
    U_del = K * (r - f);

    Console.WriteLine("Udel: " + U_del.ToString());

    double ut_Output = pastU + U_del[0];
    pastU = ut_Output;
    tempU = ut_Output;

    if (ut_Output > 10)
    {
        ut_Output = 9;
    }
    else
    {
        if (ut_Output < -10)

```



```

    {
        ut_Output = -9;
    }
}
if (ut_Output < 0.03 && ut_Output > 0)
{
    ut_Output = 0.031;
}
else
{
    if (ut_Output < 0 && ut_Output > -0.09)
    {
        ut_Output = -0.091;
    }
}

if (eNow < 0.2 && eNow > -0.2)
{
    ut_Output = 0;
}

yP = C_T * X_now;

double yP_doub = yP.ToArray()[0];
X_now = A_ss * X_now + B_ss * tempU + L * (yt_PlantOutput - yP_doub);

Console.WriteLine("Xnow: " + X_now.ToString());
Console.WriteLine("yp_Doub: " + yP_doub.ToString());

// Send output to DAQ
string ControlLoop = comboBoxControlLoop.Text;
if (ControlLoop == "OPEN")
{ ut_Output = rt_SetPoint; }

output.SetSingleValueAsVolts(0, ut_Output);
//DAQ.Output(0, ut_Output);

ut_PlantInput = ut_Output;    // Read and Display Controller Output
textBoxControllerOutput_ut.Text = ut_PlantInput.ToString("0.000");
WriteData += "," + ut_PlantInput.ToString("0.000");
chart1.Series["ut_Input"].Points.AddXY(XValue, ut_PlantInput);

```

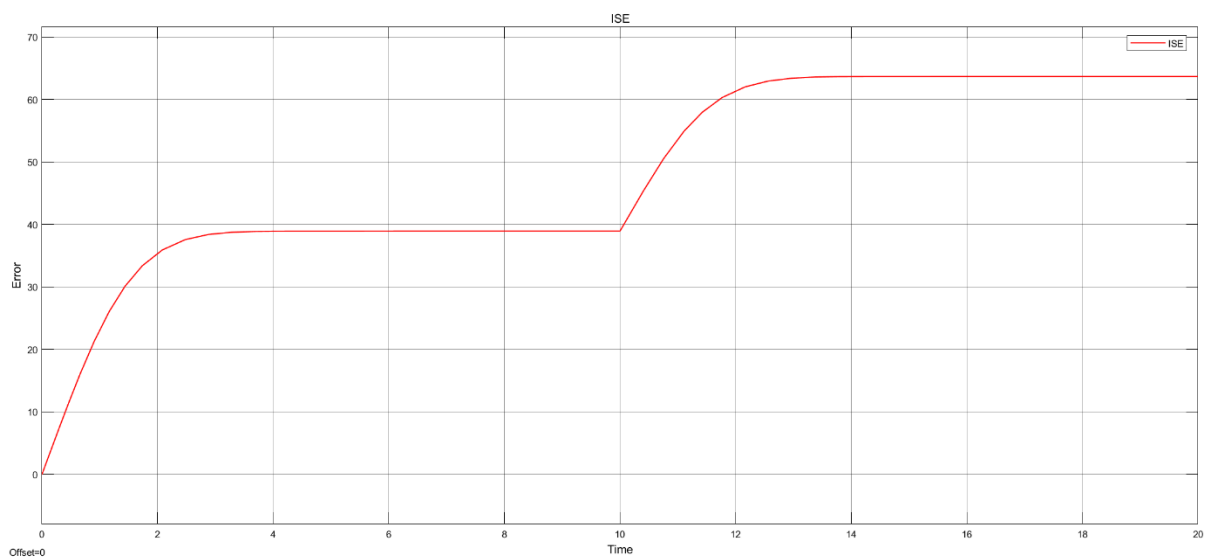


Figure 28: ISE graph for simulated state feedback controller

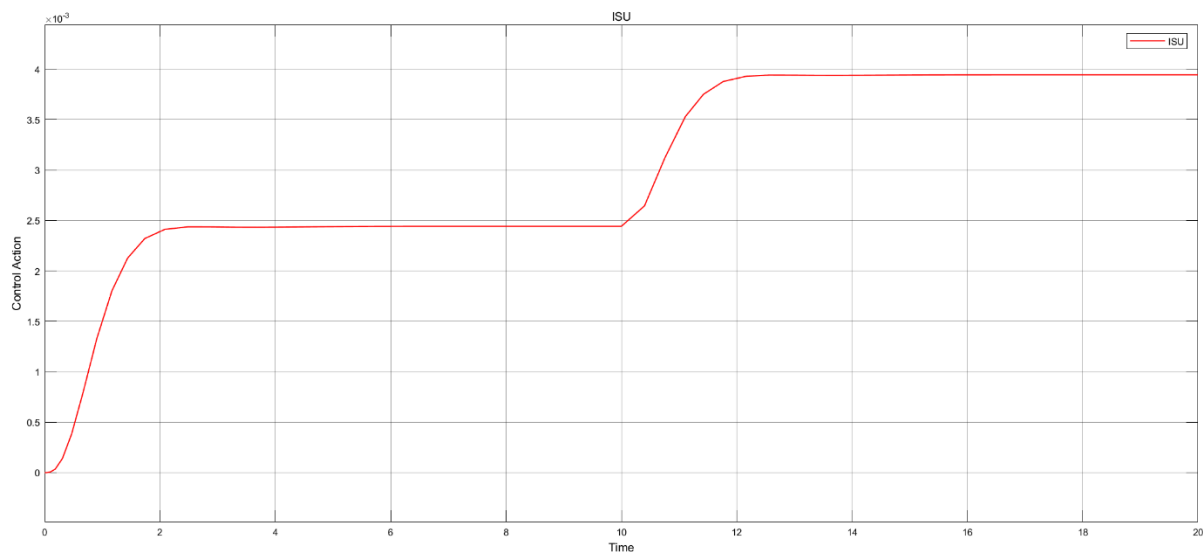


Figure 31: ISU curve for simulated state feedback controller

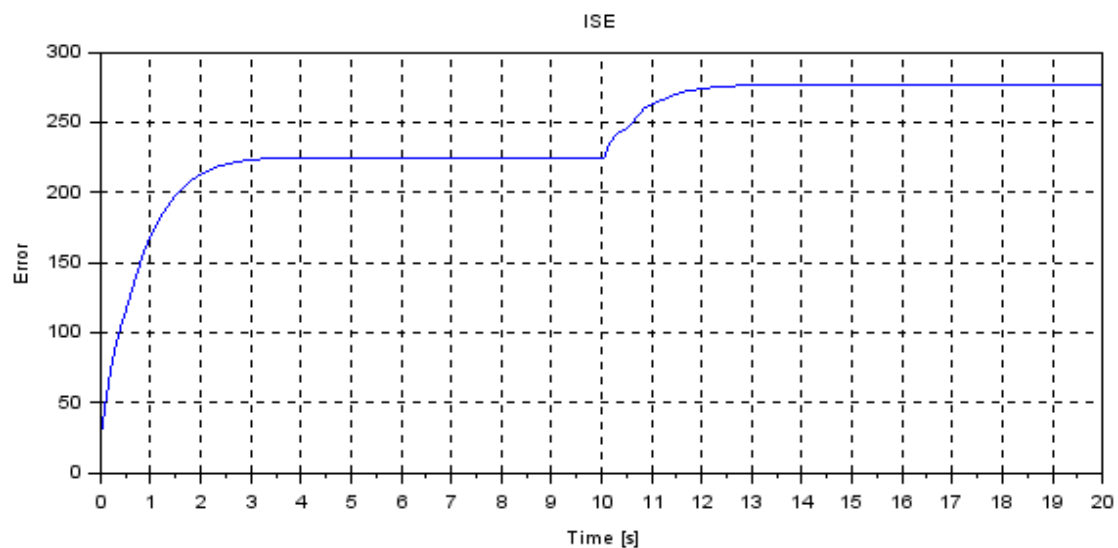


Figure 30: ISE graph for simulated MPC controller

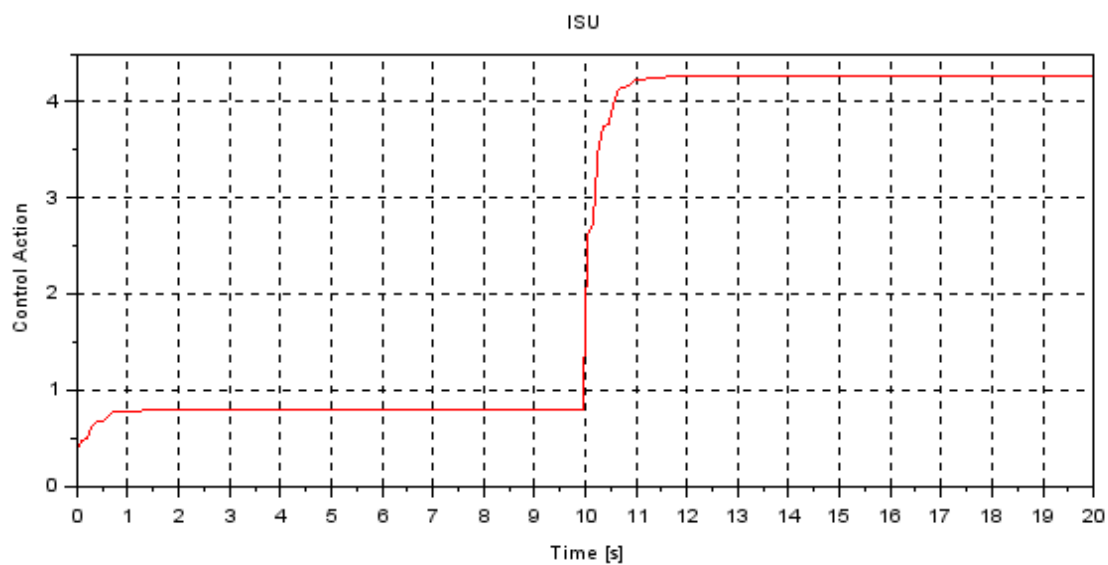


Figure 29: ISU curve for simulated MPC controller