UNIVERSITY OF WATERLOO

Faculty of Engineering
Department of Mechanical and Mechatronics Engineering

Lab 3
Ball Position Control

A report prepared for
ECE 484 - Digital Control Applications

Prepared By
Mahmubul Hoque (20462472)
Nhat Le (20416544)

Group #46
4A Mechatronics Engineering
October 31, 2017

The authors of this report declare that, in doing the lab work and writing up the lab report for ECE484, we followed rules 2, 3, 4, and 5 described at the beginning of the lab manual.

Signatures:

Mahmubul Hoque

Mahmubul Hoque

Nhat Le

Table of Contents

Table of Contents 2

Table of Figures & Tables 3

Introduction 4

Lab Procedure 5

2. (a) Simulate Lead Controller 5

2. (b) Discretize Lead Controller 5

2. (c) Design a Controller 7

2. (d) Discretize the Designed Controller 9

2. (e) 12

Appendix 15

A) Formula Node 15

B) Code 17

Table of Figures & Tables

Figure 1: General Overall Block Diagram 4

Figure 2: Simulink Graph of Input (cm) and Output (cm) vs. Time (s) using $C2LDs$ 5

Figure 3: Simulink Graph of Input (cm) and Output (cm) vs. Time (s) using $D2LDz$ 6

Figure 4: LabView Graph of Input (cm), Output (cm), and Motor Voltage (V) vs. Time (s) using $D2LDz$ 6

Figure 5: Good Region for Roots in Complex Plane 8

Figure 6: Simulink of Input (cm), Output (cm), and Motor Voltages (V) vs. Time (s) using $C2s$ 9

Figure 7: Block Diagram for Ball Control w/ Inner and Outer Loop using $C2s$ 9

Figure 8: Simulink Graph of Input (cm), Output (cm), and Motor Voltages (V) vs. Time (s) using $D2z$ 10

Figure 9: LabVIEW Graph of Input (cm), Output (cm), and Motor Voltages (V) vs. Time (s) using $D2z$ 11

Figure 10. Anti-stiction implementation 12

Figure 11. Redesigned C2 without anti-stiction block 13

Figure 12. Redesigned C2 with anti-stiction block 14

Table 1: Unaltered Values used in Previous Labs 4

Table 2: Parameters for Figure 2 5

Table 3: Compare Parameters for Figure 3&4 6

Table 4: Parameters for Figure 6 9

Table 5: Compare Parameters for Figure 8 10

Table 6: Compare Parameters for Figure 6 11

Introduction

This lab will focus on ball position control using the block diagram in Figure 1. The goal is to Ball Position, y, to follow a square wave between set values. Firstly, it will be done with the given lead Controller $C_{2LD}(s)$, which will then be discretized to run in the lab. Then, we will design our own controller $C_2(s)$, which will then be discretized to run in the lab. Finally, stiction values will be adjusted in order to improve the controller.

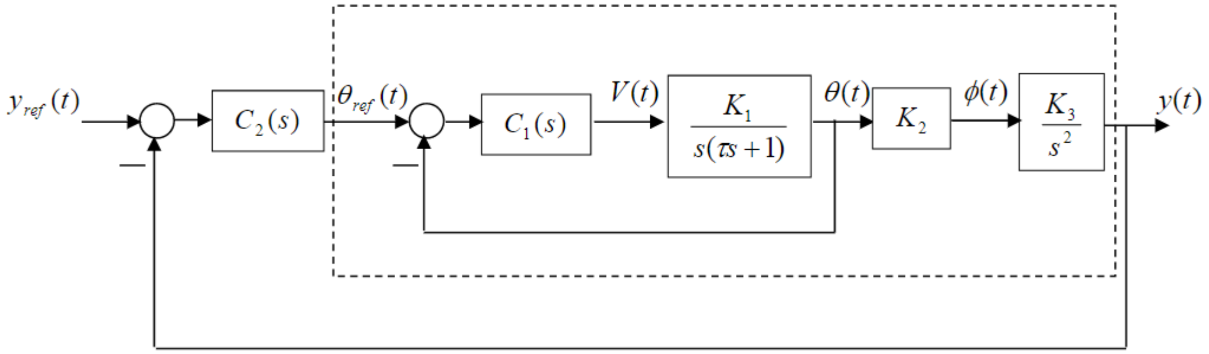


Figure 1: General Overall Block Diagram

Constants from the previous labs can be viewed in Table 1.

K_1	$\tau\left(\frac{sec}{rad}\right)$	CW Motor Stiction (V)	CCW Motor Stiction (V)	K_2	$K_3\left(\frac{m}{s^2}\right)$	Inner Loop Controller $C_1(s)$
1.6838	0.220	0.48	-0.73	0.062	4.78	$\frac{34.79s + 6681}{38.61s + 1366}$

Table 1: Unaltered Values used in Previous Labs

For the inner loop, the following scaling equation converts motor voltage to angle in radian.

$$ServoAng \; = \; -1.4241 * angV + 8.122$$

For the outer loop, the following scaling equation converts ball position voltage into ball position in centimeters. In the previous lab, it output millimetres, so we divided it by 100 to output centimeters.

$$Ball\ Position = \frac{9.605*posV - 30.278}{100}$$

Lab Procedure

2. (a) Simulate Lead Controller

$$C_{2LD}(s) = 7\frac{(s+0.35)}{s+2.5}$$

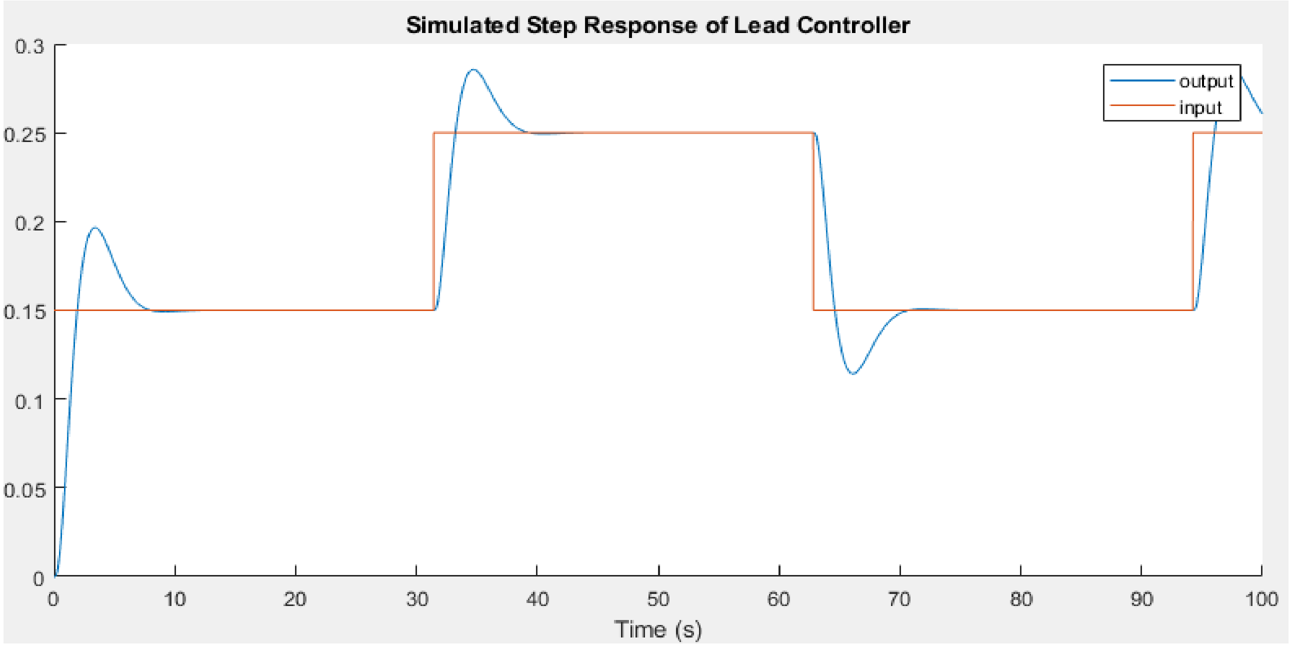


Figure 2: Simulink Graph of Input (cm) and Output (cm) vs. Time (s) using $C_{2LD}(s)$

$\%\boldsymbol{OS}$ (%)	$\boldsymbol{T_S}$ $(\boldsymbol{sec})$	$\boldsymbol{E_{SS}}$ (%)
14.28	10.14	0

Table 2: Parameters for Figure 2

2. (b) Discretize Lead Controller

Firstly, decide on a sampling period. Using Matlab, the bandwidth of the control system was found to be $f_{BW} = 2.013\ Hz$. Knowing that the sampling frequency should be 25x greater than the bandwidth…

$$T = \frac{1}{25 * f_{BW}} = 0.01987 \approx 0.02$$

To discretize, bilinear transformation is used…

$$D_{2LD}[z] = C_{2LD}(s)\Big|_{s=\frac{2}{T}\frac{z-1}{z+1}} = 7\frac{\left(\frac{2}{T}\frac{z-1}{z+1}+0.35\right)}{\frac{2}{T}\frac{z-1}{z+1}+2.5}$$

$$D_{2LD}[z] = \frac{6.99248439450687z-6.99003745318352}{z-0.997503121098627}$$

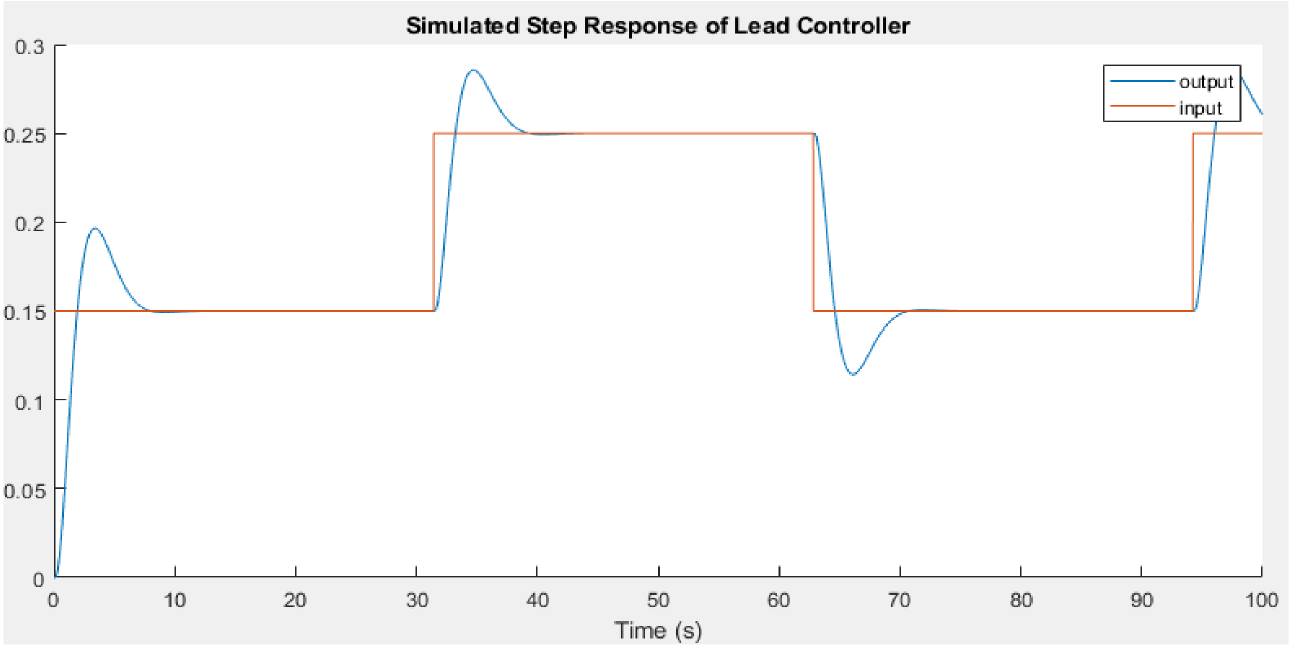


Figure 3: Simulink Graph of Input (cm) and Output (cm) vs. Time (s) using $D_{2LD}[z]$

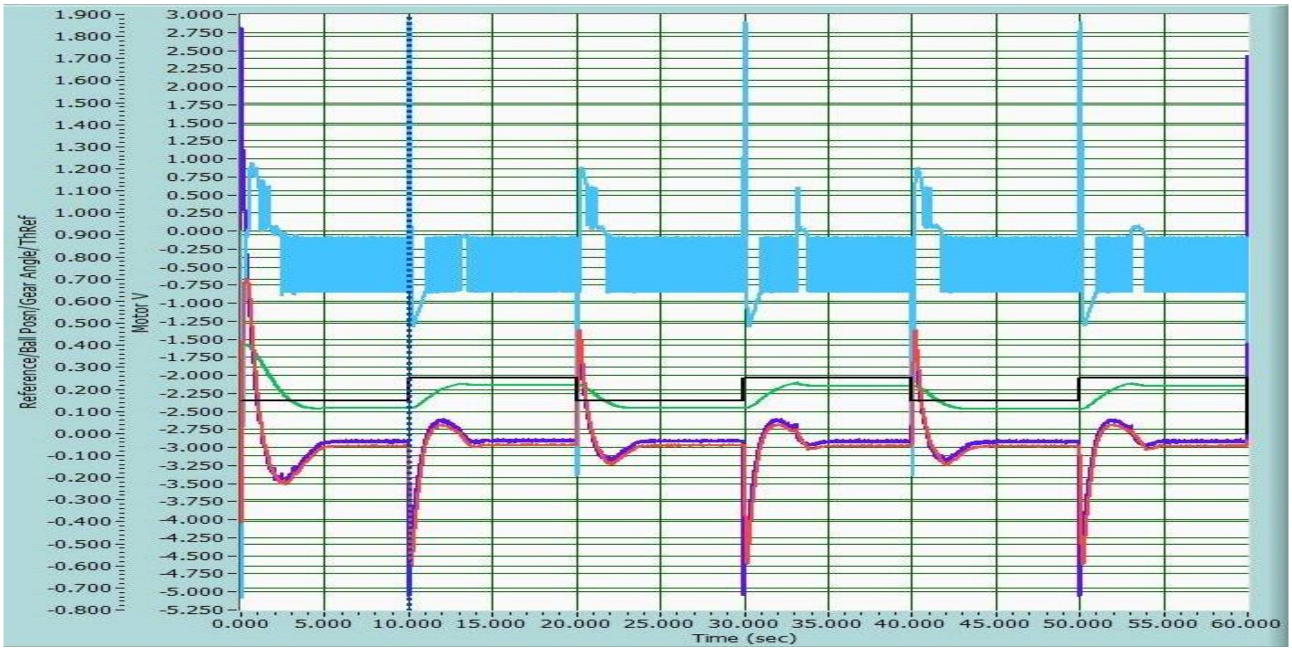


Figure 4: LabView Graph of Input (cm), Output (cm), and Motor Voltage (V) vs. Time (s) using $D_{2LD}[z]$

Figure	$\%\boldsymbol{OS}$ $(\%)$	$\boldsymbol{T_S}$ $(\boldsymbol{sec})$	$\boldsymbol{E_{SS}}$ $(\boldsymbol{cm})$
3	14.5	10.2	0
4	0.2	4	0.025

Table 3: Compare Parameters for Figure 3&4

Compare Simulink and LabVIEW Graphs

The LabVIEW graph is very different from the Simulink graph. Firstly, the $\%OS$ is almost zero in LabView and this is possible because such small roots were chosen, which would indicate a slow changing system. The settling time is also smaller in LabVIEW, possibly because there was no overshoot to begin with. Finally, Simulink had a steady state error of zero while LabVIEW had $0.025cm$; possibly because of the lack of a pure integrator, which is explained in the next paragraph.

Did we Meet Expectation?

Yes, there is symmetrical response in Figure 4; the error is consistently equal to $0.025cm$

No, steady state error is not zero. A pure integrator can possibly fix this as it would have taken the error and provided higher motor voltages overtime until the error reached zero.

2. (c) Design a Controller

Design a Controller for a square wave input from $0.15m{\sim}0.18m$

- Step response steady state error is zero
- Step response 2% $T_s < 7\ s$
- Step response $\%OS < 45\%$
- θ_{ref} does not saturate

Firstly, to design a controller, one should simplify the portions after the controller, as seen in the dashed square in Figure 1, to yield …

$$P'(s) = \left(P'_1(s)\right)\left(P'_2(s)\right) = \left(\frac{C_1\frac{K_1}{s(\tau s+1)}}{C_1\frac{K_1}{s(\tau s+1)}+1}\right) * \left(\frac{K_2K_3}{s^2}\right)$$

Within Matlab, when looking at individual bandwidths one will notice that $f_{BW\_P'_1} \gg f_{BW\_P'_2}$. As such, $P'_1(s)$ can be simplified to a DC Gain of 1.

$$P'_1(s) = 1 \rightarrow P'(s) = 1 * P'_2 = \frac{K_2K_3}{s^2}$$

Now when including the integrator, one will yield the Augmented Plant…

$$P_{aug} = P'_1(s) * P'_2(s) * \frac{1}{s} = \frac{K_2K_3}{s^3}$$

Since the Augmented Plant is of order 3, the new controller will be of order 2, as represented by…

$$C_2(s) = \frac{n_2 s^2 + n_1 s^1 + n_0}{d_2 s^2 + d_1 s^1 + d_0}$$

We know we need to find 5 roots because $\#Roots\ =\ n_{plant} + n_{ctrl}$; however, to find those roots, one needs to find the damping ratio ζ and natural frequency ω_n first, using the following equations…

$$\zeta = \frac{-\ln(\%OS)}{\sqrt{\pi^2 + (\ln(\%OS))^2}} = 0.246$$

$$\omega_n = \frac{4}{T_S}|\zeta| = 0.141$$

One can then find the limiting range of roots, using the previous values with the following equations…

$$\theta = \cos^{-1}(\zeta) = 75.76°$$

$$xShift = \zeta\omega_n = 0.0345$$

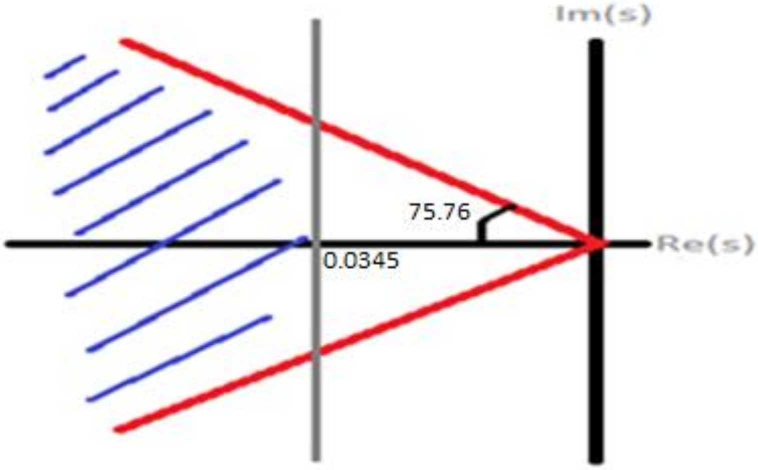


Figure 5: Good Region for Roots in Complex Plane

As such, using Figure 5, the selected roots are…

$$\Lambda = \{-0.65, -0.65, -0.65, -0.65, -2\}$$

To ease the math required, we used the PP() function provided on Learn. Also, since the Matlab block diagram does not contain $P_{aug}(s)$, the integrator $\frac{1}{s}$ was added to $C_2(s)$, as follows…

$$C_2(s) = pp(P'(s), \Lambda) * \frac{1}{s} = \frac{20.31s^2 + 7.821s + 1.175}{s^3 + 4.6s^2 + 7.735s}$$

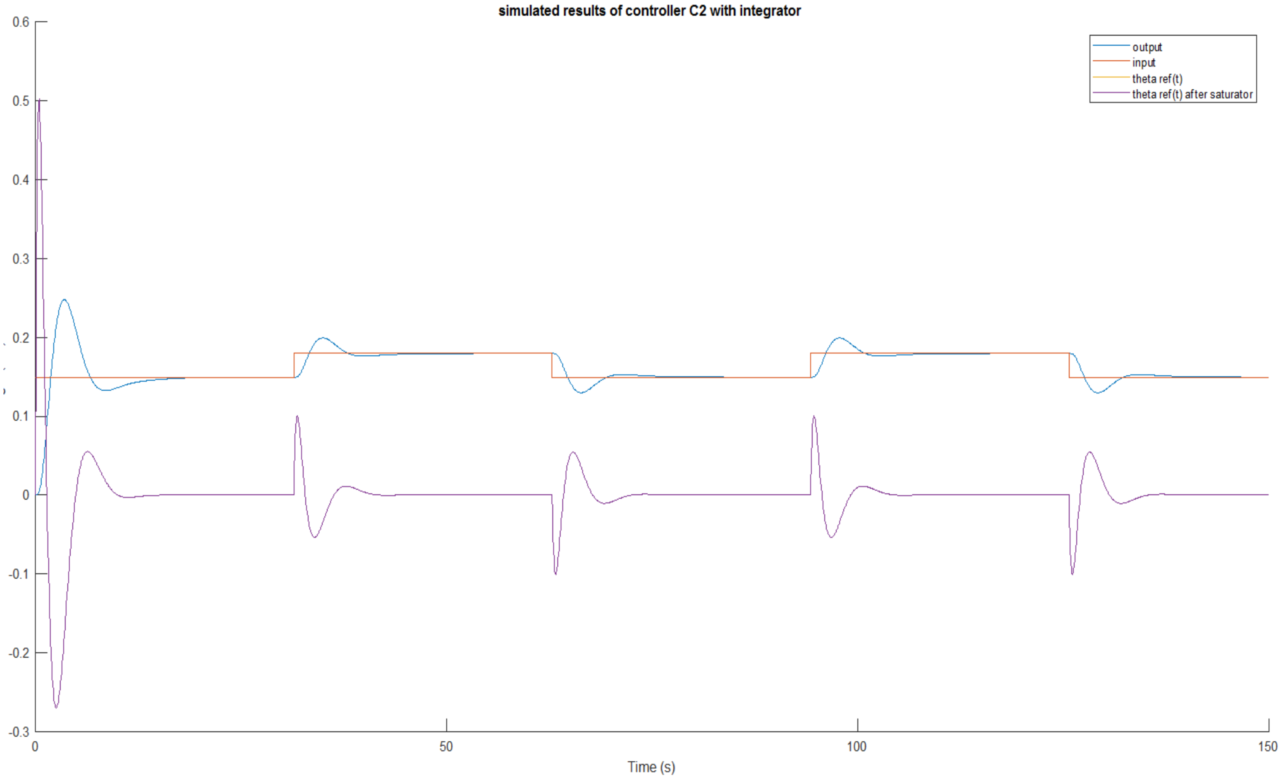


Figure 6: Simulink of Input (cm), Output (cm), and Motor Voltages (V) vs. Time (s) using $C_2(s)$

$\%\boldsymbol{OS}\ (\%)$	$\boldsymbol{T_S}\ (\boldsymbol{sec})$	$\boldsymbol{E_{SS}}\ (\boldsymbol{cm})$	$\boldsymbol{\theta_{ref}}$
11	6.6	0	$-0.7 < \theta_{ref} < 0.7$

Table 4: Parameters for Figure 6

As seen in Figure 6 and Table 4, $C_2(s)$ meets all specifications. Overshoot is less than 45%, settling time is less than 7 seconds, steady state error is zero, and motor voltage does not saturate.

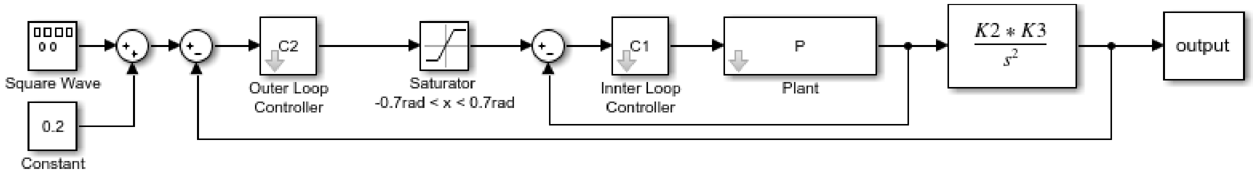


Figure 7: Block Diagram for Ball Control w/ Inner and Outer Loop using $C_2(s)$

2. (d) Discretize the Designed Controller

Firstly, decide on a sampling period. Using Matlab, the bandwidth of the control system was found to be $f_{BW} = 1.844\ Hz$. Knowing that the sampling frequency should be 25x greater than the bandwidth…

$$T = \frac{1}{25 * f_{BW}} = 0.0217 \approx 0.02$$

Using Bilinear Transformation …

$$D_2[z] = C_2(s)\big|_{s=\frac{2}{T}\frac{z-1}{z+1}} = \frac{n_2\left(\frac{2}{T}\frac{z-1}{z+1}\right)^2 + n_1\left(\frac{2}{T}\frac{z-1}{z+1}\right)^1 + n_0}{d_2\left(\frac{2}{T}\frac{z-1}{z+1}\right)^2 + d_1\left(\frac{2}{T}\frac{z-1}{z+1}\right)^1 + d_0}$$

$$D_2[z] = \frac{0.01013z^3 - 0.01013z^2 - 0.01013z + 0.01013}{z^3 - 2.995z^2 + 2.991z - 0.9954}$$

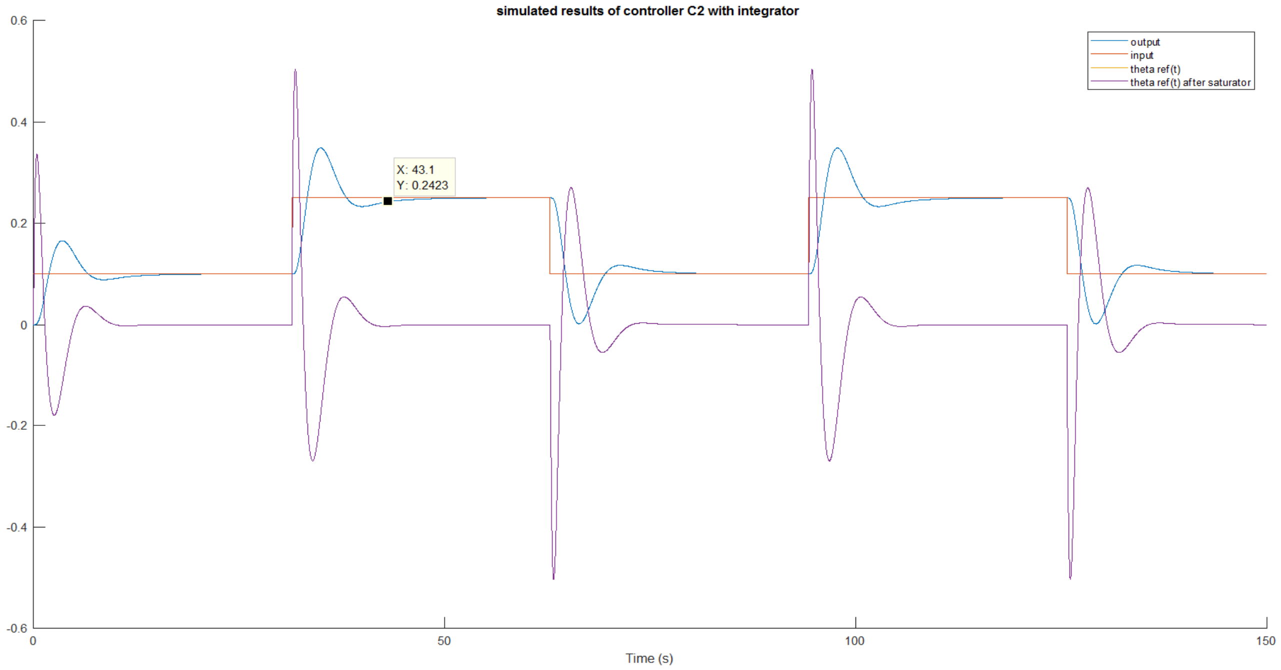


Figure 8: Simulink Graph of Input (cm), Output (cm), and Motor Voltages (V) vs. Time (s) using $D_2[z]$

$\%\boldsymbol{OS}$ (%)	$\boldsymbol{T_S}$ $(\boldsymbol{sec})$	$\boldsymbol{E_{SS}}$ $(\boldsymbol{cm})$	$\boldsymbol{\theta_{ref}}$
39.2	11	0	$-0.7 < \theta_{ref} < 0.7$

Table 5: Compare Parameters for Figure 8

As one can see from Figure 8 and Table 3, the discrete controller only meets ¾ of the parameters. Settling time is greater than 7s. We were not able to meet all the specifications no matter how many roots we tried. Experimental analysis taught us:

- As roots increased in magnitude (for ex. {-10, -10, -10, -10, -10})
    - T_s decreased, but θ_{ref} would saturate because bigger roots -> faster system
    - θ_{ref} tends to saturate because greater input range -> motor working harder
- As roots decreased in magnitude (for ex. {-0.6, -0.6, -0.6, -0.6, -0.6})
    - θ_{ref} did not saturate, but T_s increased because smaller roots -> slower system
    - Overshoot decreases, sometimes to zero
    - Error increased because there is little to no overshoot

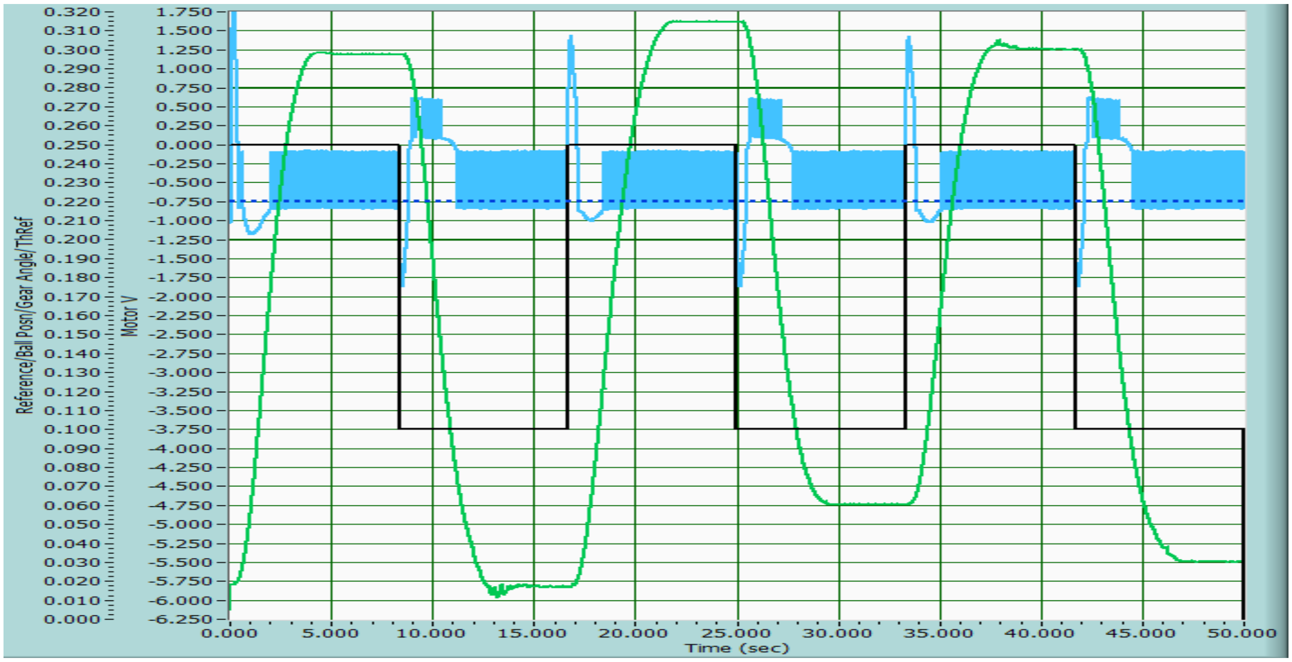


Figure 9: LabVIEW Graph of Input (cm), Output (cm), and Motor Voltages (V) vs. Time (s) using $D_2[z]$

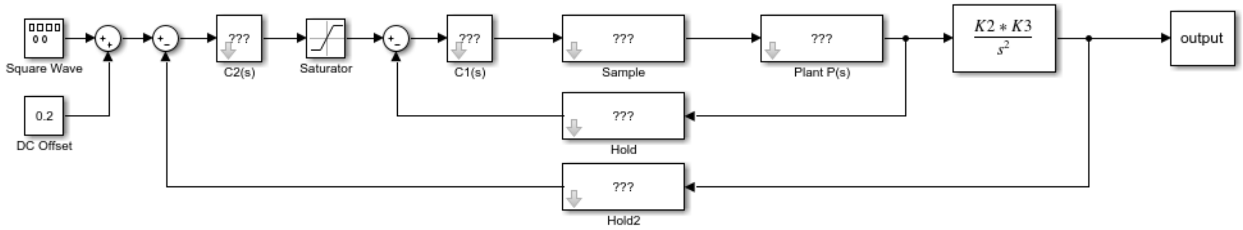
$\%\boldsymbol{OS}$ (%)	$\boldsymbol{T_S}$ ($\boldsymbol{sec}$)	$\boldsymbol{E_{SS}}$ ($\boldsymbol{cm}$)	$\boldsymbol{\theta_{ref}}$
1.3	5	0.05	$-0.7 < \theta_{ref} < 0.7$

Table 6: Compare Parameters for Figure 6

As one can see from Figure 8 and Table 6, the LabVIEW outputs a graph that is very different from Simulink. Both overshoot and motor voltage are within limits; however, in the LabVIEW graph, settling time met the specification, but steady state error did not. The steady state error was not able to reach zero for a few possible reasons:

- Overshoot too small for controller to compensate
- Stiction values required changing to allow for the small changes in error to be accommodated

Moreover, note how the pure integrator is influencing the system. Overtime, the error is decreasing.



2. (e) Stiction and New Controller

Part e deals with the ball stiction that plagues the system. As such, it's imperative that an anti-stiction block be added to fix this error. The block can be seen in Figure 10 below.

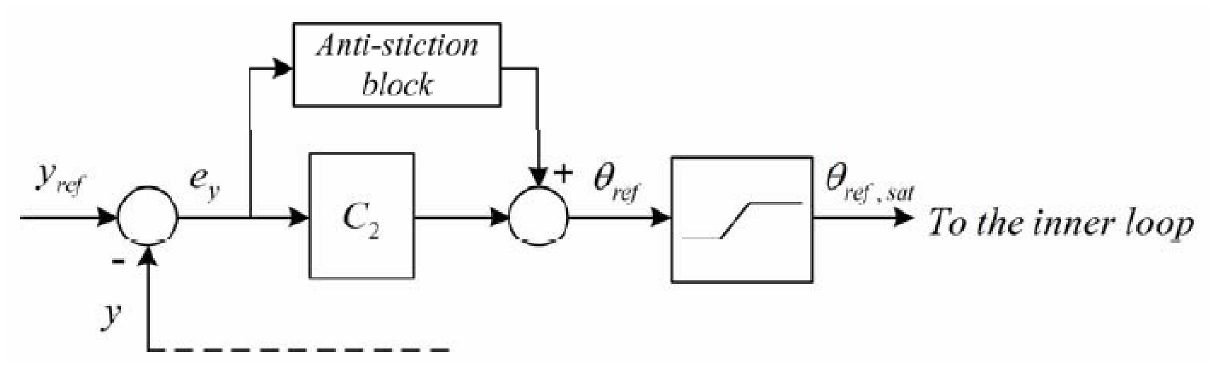


Figure 10. Anti-stiction implementation

Along with this anti-stiction block, the controller C_2 will also be changed to have the following specifications:

- Linearized feedback system is stable
- 4% step response steady-state error
- 2% step response settling time $\leq 10s$
- $\leq 45\%$ step response overshoot

From looking at part d, it can be seen that the previously designed controller in part c will not be sufficient to meet all the specifications above. As such, the controller from part d is looked at, which shows that the results will be able to meet the specifications above with some modifications. Due to the fact that an integrator is needed to fix the steady-state error over time, an integrator is added to the controller in part a. Due to this, a zero is needed, which will form the following controller equation:

$$C_2 = 7\frac{s+0.35}{s+2.5} \times \frac{1}{s} \times (s-x)$$

Where x represents the zero. After several iterations of trial and error, the final chosen zero has a value of $x = -0.01$. As such, the controller is:

$$C_2 = 7\frac{s+0.35}{s+2.5} \times \frac{1}{s} \times (s+0.01) = \frac{7s^2 + 2.52s + 0.0245}{s^2 + 2.5s}$$

This controller is now discretized by using a similar method as from part d, with the same sample time value. The discretized controller is:

$$C_{2D} = \frac{6.993z^2 - 13.98z + 6.99}{z^2 - 1.998z + 0.9975}$$

This controller is then imported into LabView, following a similar method as from previous parts. The experiment is then run with a 0.10m to 0.25m square wave input, and the result can be seen in Figure 11 below.

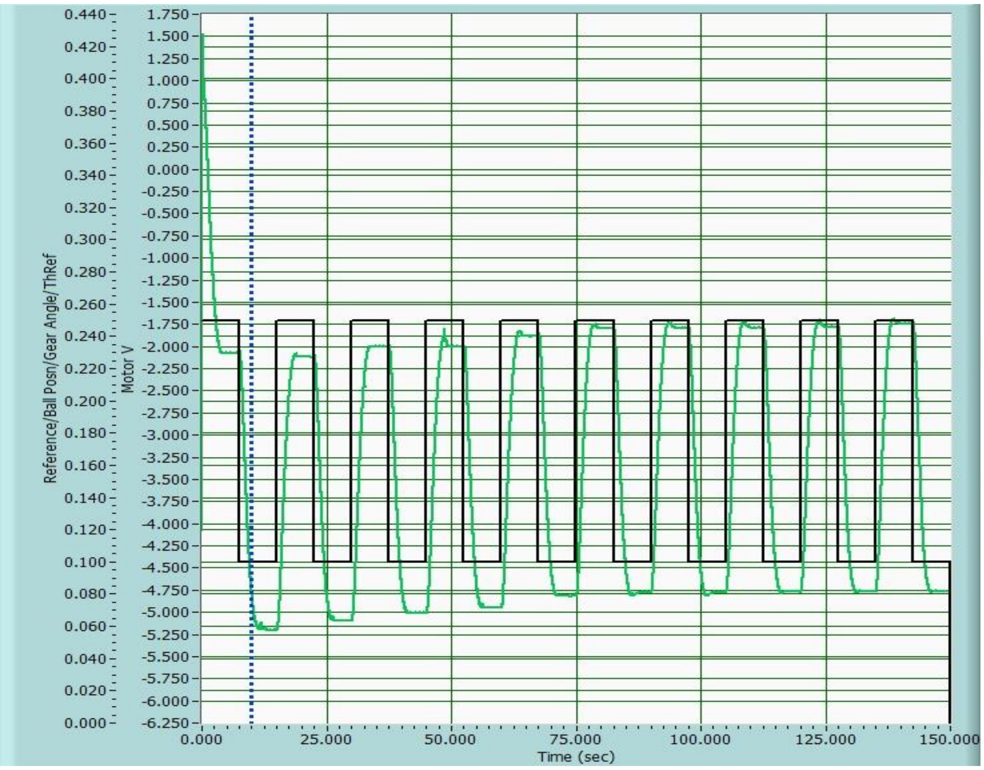


Figure 11. Redesigned C2 without anti-stiction block

As seen from Figure 11 above, it can be seen that without the stiction block, as more cycles are completed, the steady state values are close but not within the limit for the controller specifications above. The anti-stiction block is now completed by doing iterations, each changing the ThRef values. The anti-stiction block code that was implemented into LabView can be seen below.

```
if (eb < 0) {
   if (eb < -0.02) {
      refSaturated = refSaturated - 0.1;
   } else if (eb > -0.02) {
      refSaturated = refSaturated - 0.07;
   }
}
if (eb > 0) {
   if (eb > 0.02) {
      refSaturated = refSaturated + 0.1;
   }
}
```

The experiment is then rerun, and the output can be seen in Figure 12 below.

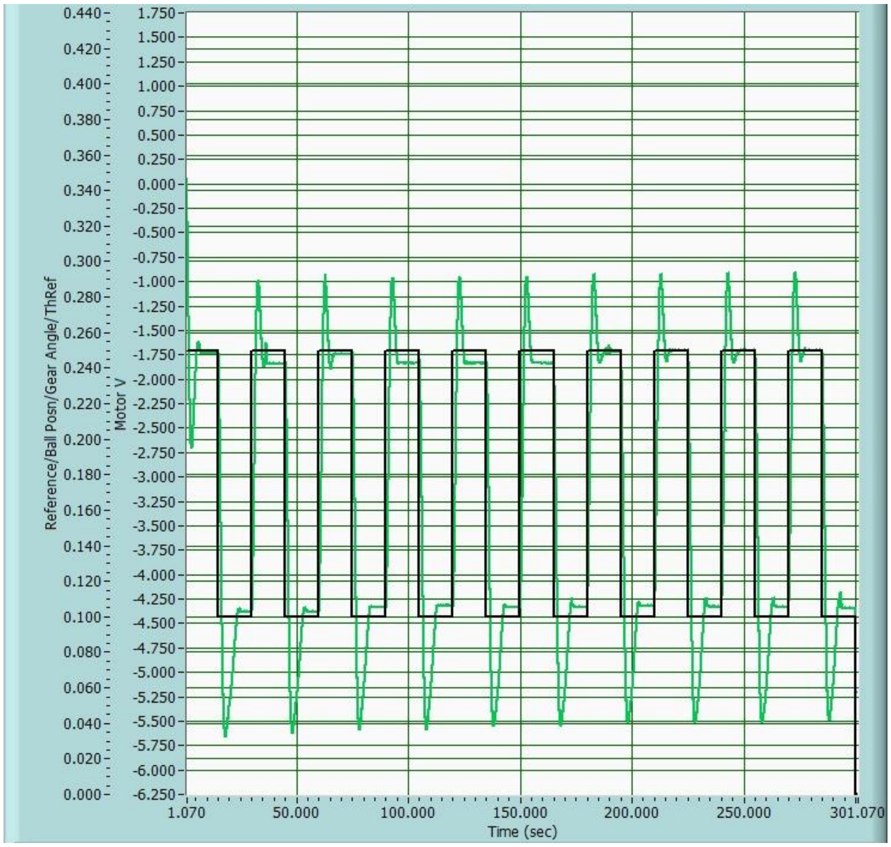


Figure 12. Redesigned C2 with anti-stiction block

It can be seen that with this implementation and the anti-stiction block, the specifications for C_2 have been met. The values associated with the plot were read off the plots, and listed below.

$$\%OS = \frac{M_P - SS}{SS} \times 100 = \frac{0.285 - 0.25}{0.25} \times 100 = 14\%$$

$$T_{settling} = 6s$$

It can also be seen that the steady state values of the plots are 0.105m when the input's 0.10m, and 0.245m when the input's 0.25m. Hence, the steady-state error is:

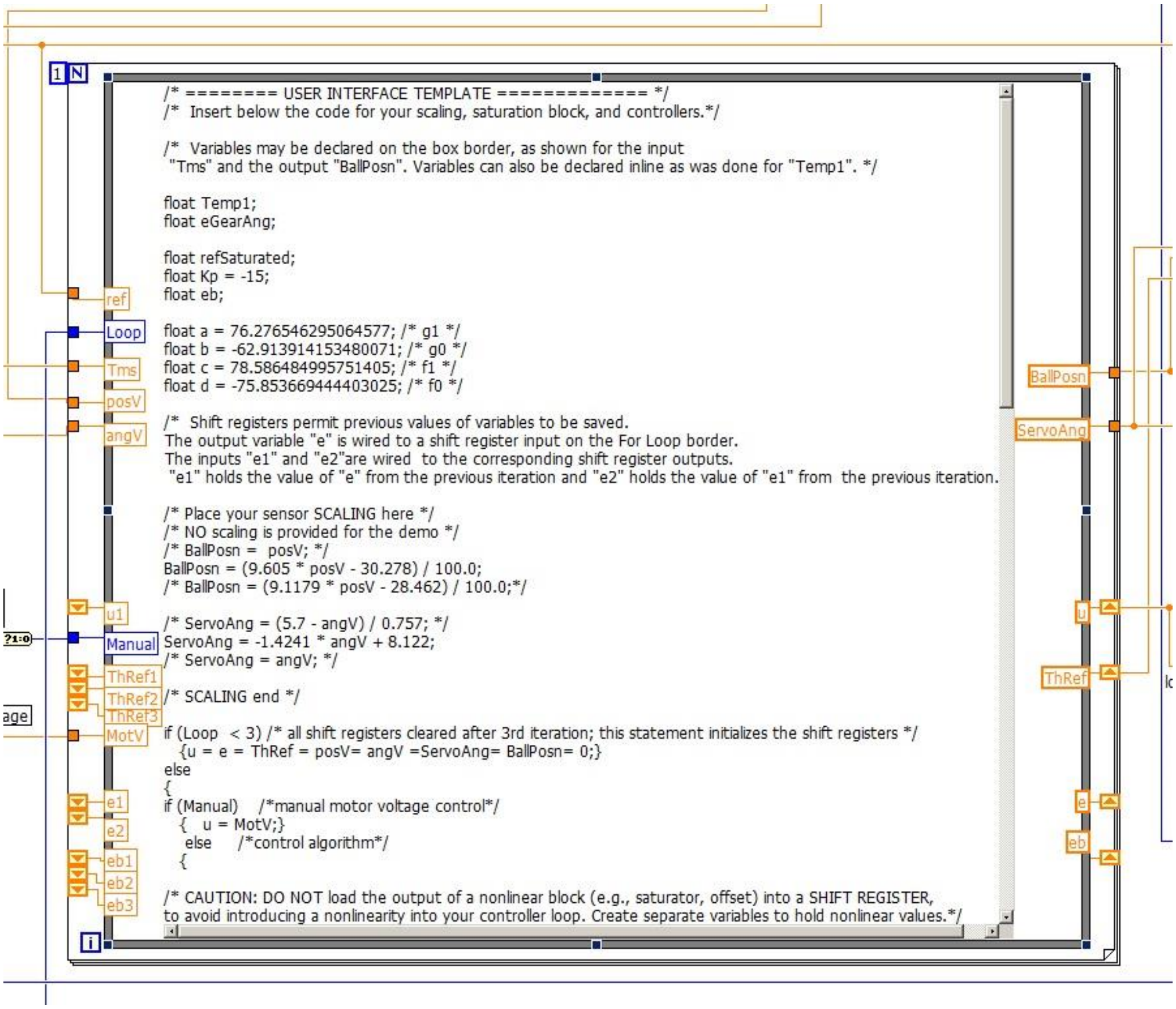
$$SS_{error_{0.10m}} = \frac{0.103 - 0.1}{0.1} \times 100 = 3\%$$

$$SS_{error_{0.25m}} = \frac{0.245 - 0.25}{0.25} \times 100 = 2\%$$

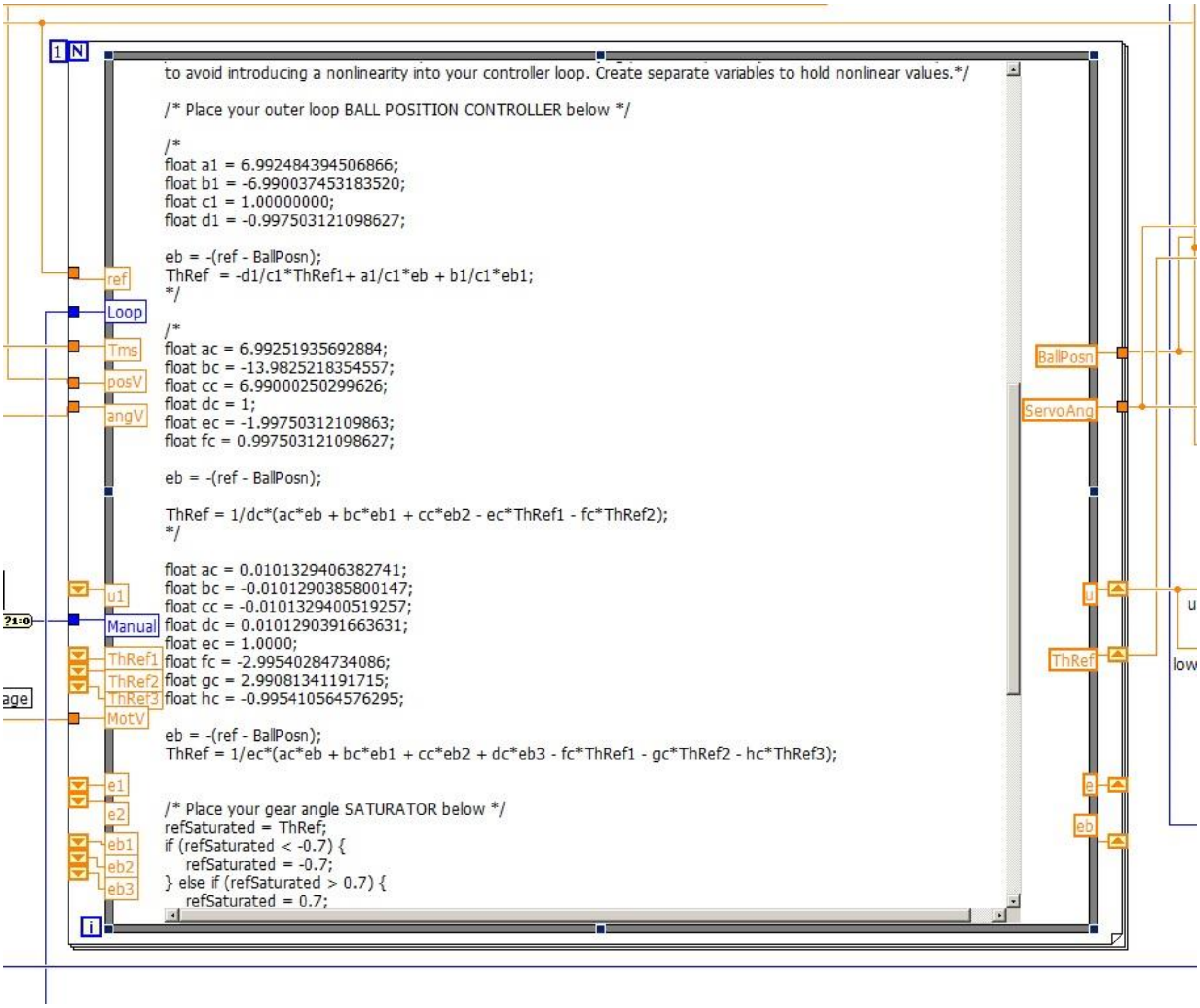
Hence, the specifications above have been met.

Appendix

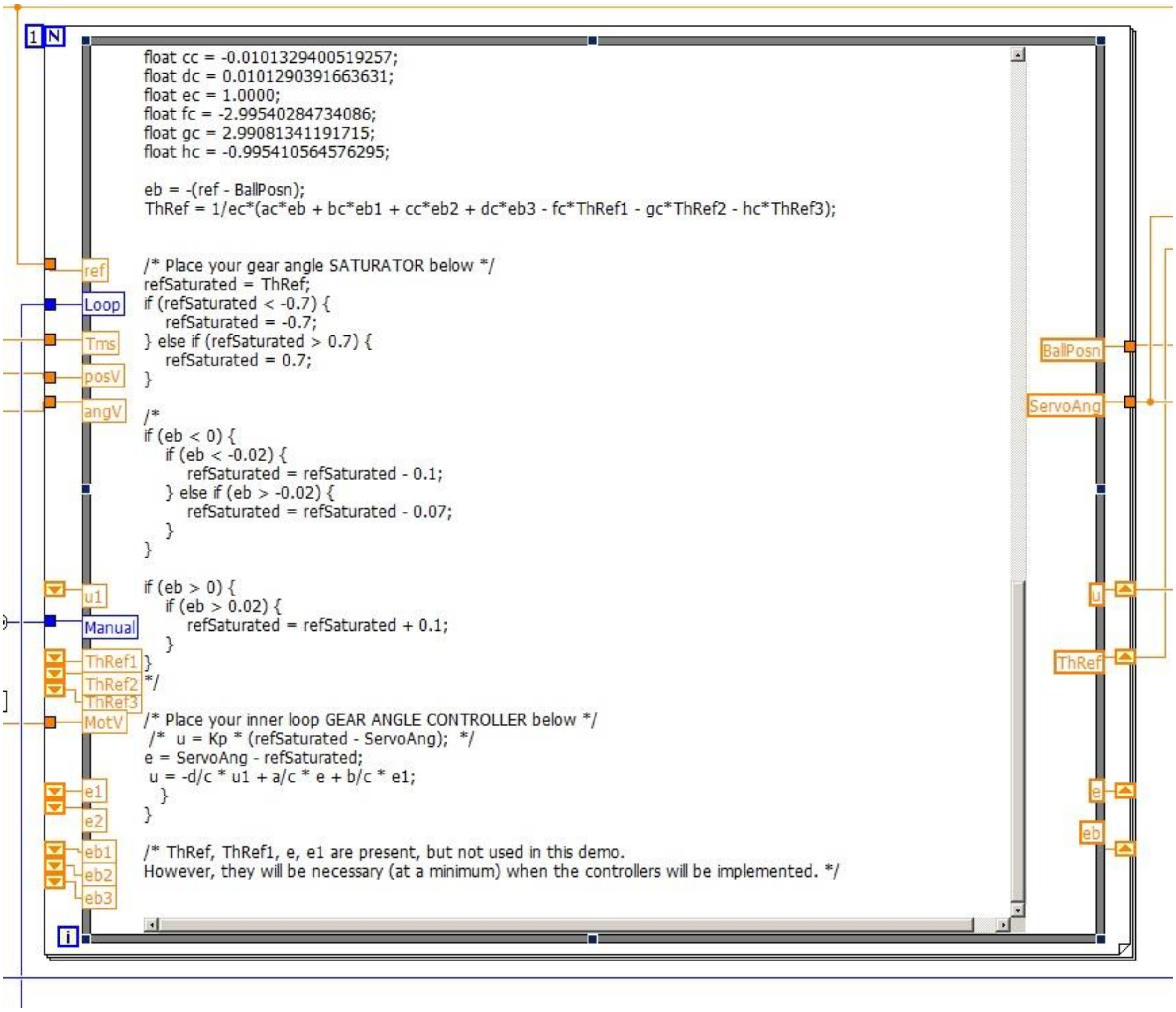
A) Formula Node



```
/* ======== USER INTERFACE TEMPLATE ============= */
/*  Insert below the code for your scaling, saturation block, and controllers.*/

/*  Variables may be declared on the box border, as shown for the input
 "Tms" and the output "BallPosn". Variables can also be declared inline as was done for "Temp1". */

float Temp1;
float eGearAng;

float refSaturated;
float Kp = -15;
float eb;

float a = 76.276546295064577; /* g1 */
float b = -62.913914153480071; /* g0 */
float c = 78.586484995751405; /* f1 */
float d = -75.853669444403025; /* f0 */

/*  Shift registers permit previous values of variables to be saved.
The output variable "e" is wired to a shift register input on the For Loop border.
The inputs "e1" and "e2"are wired  to the corresponding shift register outputs.
 "e1" holds the value of "e" from the previous iteration and "e2" holds the value of "e1" from  the previous iteration.

/* Place your sensor SCALING here */
/* NO scaling is provided for the demo */
/* BallPosn =  posV; */
BallPosn = (9.605 * posV - 30.278) / 100.0;
/* BallPosn = (9.1179 * posV - 28.462) / 100.0;*/

/* ServoAng = (5.7 - angV) / 0.757; */
ServoAng = -1.4241 * angV + 8.122;
/* ServoAng = angV; */

/* SCALING end */

if (Loop  < 3) /* all shift registers cleared after 3rd iteration; this statement initializes the shift registers */
   {u = e = ThRef = posV= angV =ServoAng= BallPosn= 0;}
else
{
if (Manual)    /*manual motor voltage control*/
   {   u = MotV;}
    else     /*control algorithm*/
   {

/* CAUTION: DO NOT load the output of a nonlinear block (e.g., saturator, offset) into a SHIFT REGISTER,
to avoid introducing a nonlinearity into your controller loop. Create separate variables to hold nonlinear values.*/
```



```
to avoid introducing a nonlinearity into your controller loop. Create separate variables to hold nonlinear values.*/

/* Place your outer loop BALL POSITION CONTROLLER below */

/*
float a1 = 6.992484394506866;
float b1 = -6.990037453183520;
float c1 = 1.00000000;
float d1 = -0.997503121098627;

eb = -(ref - BallPosn);
ThRef  = -d1/c1*ThRef1+ a1/c1*eb + b1/c1*eb1;
*/

/*
float ac = 6.99251935692884;
float bc = -13.9825218354557;
float cc = 6.99000250299626;
float dc = 1;
float ec = -1.99750312109863;
float fc = 0.997503121098627;

eb = -(ref - BallPosn);

ThRef = 1/dc*(ac*eb + bc*eb1 + cc*eb2 - ec*ThRef1 - fc*ThRef2);
*/

float ac = 0.0101329406382741;
float bc = -0.0101290385800147;
float cc = -0.0101329400519257;
float dc = 0.0101290391663631;
float ec = 1.0000;
float fc = -2.99540284734086;
float gc = 2.99081341191715;
float hc = -0.995410564576295;

eb = -(ref - BallPosn);
ThRef = 1/ec*(ac*eb + bc*eb1 + cc*eb2 + dc*eb3 - fc*ThRef1 - gc*ThRef2 - hc*ThRef3);


/* Place your gear angle SATURATOR below */
refSaturated = ThRef;
if (refSaturated < -0.7) {
    refSaturated = -0.7;
} else if (refSaturated > 0.7) {
    refSaturated = 0.7;
```



```
float cc = -0.0101329400519257;
float dc = 0.0101290391663631;
float ec = 1.0000;
float fc = -2.99540284734086;
float gc = 2.99081341191715;
float hc = -0.995410564576295;

eb = -(ref - BallPosn);
ThRef = 1/ec*(ac*eb + bc*eb1 + cc*eb2 + dc*eb3 - fc*ThRef1 - gc*ThRef2 - hc*ThRef3);


/* Place your gear angle SATURATOR below */
refSaturated = ThRef;
if (refSaturated < -0.7) {
   refSaturated = -0.7;
} else if (refSaturated > 0.7) {
   refSaturated = 0.7;
}

/*
if (eb < 0) {
   if (eb < -0.02) {
      refSaturated = refSaturated - 0.1;
   } else if (eb > -0.02) {
      refSaturated = refSaturated - 0.07;
   }
}

if (eb > 0) {
   if (eb > 0.02) {
      refSaturated = refSaturated + 0.1;
   }
}
*/

/* Place your inner loop GEAR ANGLE CONTROLLER below */
 /*  u = Kp * (refSaturated - ServoAng);  */
e = ServoAng - refSaturated;
 u = -d/c * u1 + a/c * e + b/c * e1;
  }
}

/* ThRef, ThRef1, e, e1 are present, but not used in this demo.
However, they will be necessary (at a minimum) when the controllers will be implemented. */
```

B) Code

```
/* ======== USER INTERFACE TEMPLATE ============ */
/*  Insert below the code for your scaling, saturation block, and controllers.*/

/*  Variables may be declared on the box border, as shown for the input
 "Tms" and the output "BallPosn". Variables can also be declared inline as was done for "Temp1". */

float Temp1;
float eGearAng;

float refSaturated;
float Kp = -15;
float eb;

float a = 76.276546295064577; /* g1 */
float b = -62.913914153480071; /* g0 */
float c = 78.586484995751405; /* f1 */
float d = -75.853669444403025; /* f0 */

/*  Shift registers permit previous values of variables to be saved.
The output variable "e" is wired to a shift register input on the For Loop border.
The inputs "e1" and "e2"are wired  to the corresponding shift register outputs.
 "e1" holds the value of "e" from the previous iteration and "e2" holds the value of "e1" from  the
previous iteration. */

/* Place your sensor SCALING here */
/* NO scaling is provided for the demo */
/* BallPosn =  posV; */
BallPosn = (9.605 * posV - 30.278) / 100.0;
/* BallPosn = (9.1179 * posV - 28.462) / 100.0;*/

/* ServoAng = (5.7 - angV) / 0.757; */
ServoAng = -1.4241 * angV + 8.122;
/* ServoAng = angV; */

/* SCALING end */

if (Loop  < 3) /* all shift registers cleared after 3rd iteration; this statement initializes the shift
registers */
   {u = e = ThRef = posV= angV =ServoAng= BallPosn= 0;}
else
{
if (Manual)    /*manual motor voltage control*/
   {   u = MotV;}
    else     /*control algorithm*/
   {

/* CAUTION: DO NOT load the output of a nonlinear block (e.g., saturator, offset) into a SHIFT
REGISTER,
to avoid introducing a nonlinearity into your controller loop. Create separate variables to hold
nonlinear values.*/

/* Place your outer loop BALL POSITION CONTROLLER below */

/*
float a1 = 6.992484394506866;
float b1 = -6.990037453183520;
float c1 = 1.00000000;
float d1 = -0.997503121098627;

eb = -(ref - BallPosn);
ThRef  = -d1/c1*ThRef1+ a1/c1*eb + b1/c1*eb1;
*/
```

```
float ac = 6.99251935692884;
float bc = -13.9825218354557;
float cc = 6.99000250299626;
float dc = 1;
float ec = -1.99750312109863;
float fc = 0.997503121098627;

eb = -(ref - BallPosn);

ThRef = 1/dc*(ac*eb + bc*eb1 + cc*eb2 - ec*ThRef1 - fc*ThRef2);


/*
float ac = 0.0101329406382741;
float bc = -0.0101290385800147;
float cc = -0.0101329400519257;
float dc = 0.0101290391663631;
float ec = 1.0000;
float fc = -2.99540284734086;
float gc = 2.99081341191715;
float hc = -0.995410564576295;

eb = -(ref - BallPosn);
ThRef = 1/ec*(ac*eb + bc*eb1 + cc*eb2 + dc*eb3 - fc*ThRef1 - gc*ThRef2 - hc*ThRef3);
*/

/* Place your gear angle SATURATOR below */
refSaturated = ThRef;
if (refSaturated < -0.7) {
   refSaturated = -0.7;
} else if (refSaturated > 0.7) {
   refSaturated = 0.7;
}

if (eb < 0) {
   if (eb < -0.02) {
      refSaturated = refSaturated - 0.1;
   } else if (eb > -0.02) {
      refSaturated = refSaturated - 0.07;
   }
}

if (eb > 0) {
   if (eb > 0.02) {
      refSaturated = refSaturated + 0.1;
   }
}

/* Place your inner loop GEAR ANGLE CONTROLLER below */
 /*  u = Kp * (refSaturated - ServoAng);  */
e = ServoAng - refSaturated;
 u = -d/c * u1 + a/c * e + b/c * e1;
   }
}

/* ThRef, ThRef1, e, e1 are present, but not used in this demo.
However, they will be necessary (at a minimum) when the controllers will be implemented. */
```