Magnetic Suspension System Control Using Position and Current Feedback

Senior Design Project

Team: Gary Boline and Andrew Michalets

Advisors: Dr. Winfred Anakwa and Dr. Donald Schertz

Date: May 17, 2007

Bradley University Department of Electrical and Computer Engineering

Abstract

The original objective of the project was to design a controller using current and position feedback to suspend a metallic ball with an electromagnet. It was expected that the relationship between the input voltage and electromagnet current could be modeled by a linear differential equation. System identification experiments confirmed that the input voltage and electromagnet current relationship was a rate limiter. Consequently, it was not possible to utilize classical control techniques to design a controller using electromagnet current feedback. A classical digital controller, which does not include current feedback, was designed and tested using Simulink and xPC Targetbox and implemented on a Motorola ColdFire microcontroller.

Table of Contents

Summary

Magnetic suspension systems are increasingly used in industrial rotating machinery applications. They offer a number of practical advantages such as low energy consumption, capacity for linear displacement, high rotational speeds and can operate at extreme temperatures with a longer lifespan. The absence of mechanical contacts that are present in traditional systems eliminates the problem of lubrication. The Magnetic Suspension System uses an electro-magnetic force to suspend a hollow metal ball. There are two initial inputs to the system: set point and reference input. The set point is the operating point of the system, around which a reference signal is tracked.

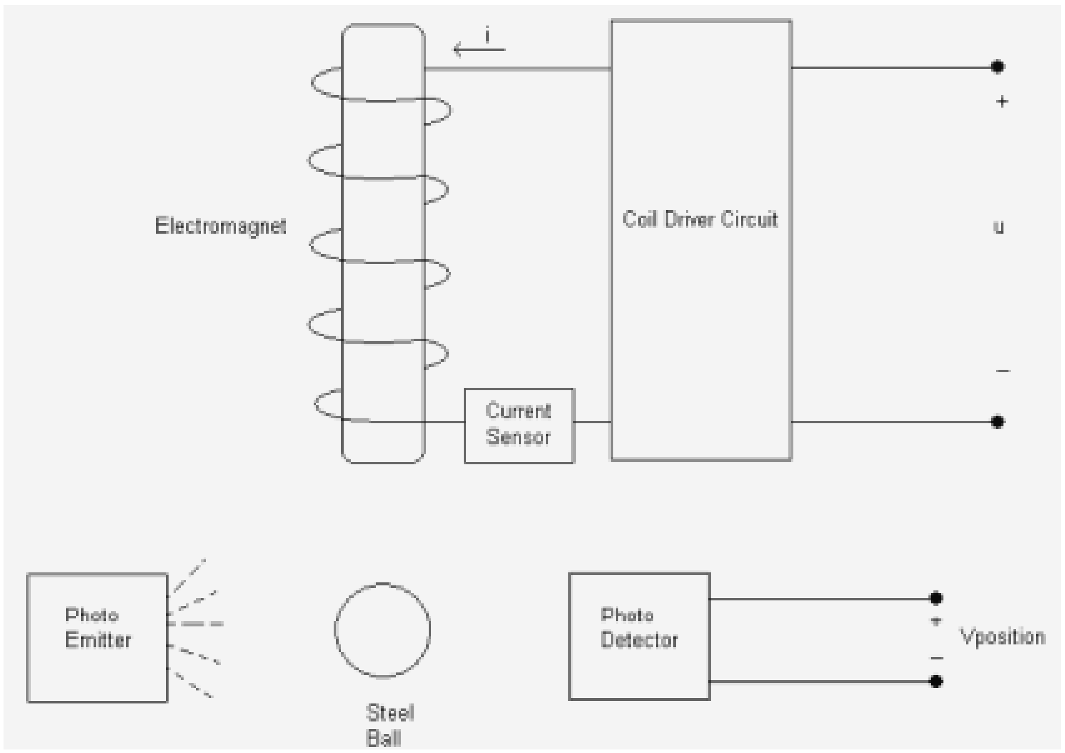


Figure 4-1: Magnetic Suspension Plant

Goals

Unlike previous controller designs using this magnetic suspension platform, a controller with current feedback along with conventional position feedback will be investigated for improved system performance. After successful completion of designing a controller, the controller will be implemented on an xPC TargetBox and a Motorola ColdFire microcontroller. The magnetic suspension system diagram is shown in Figure 5-1.

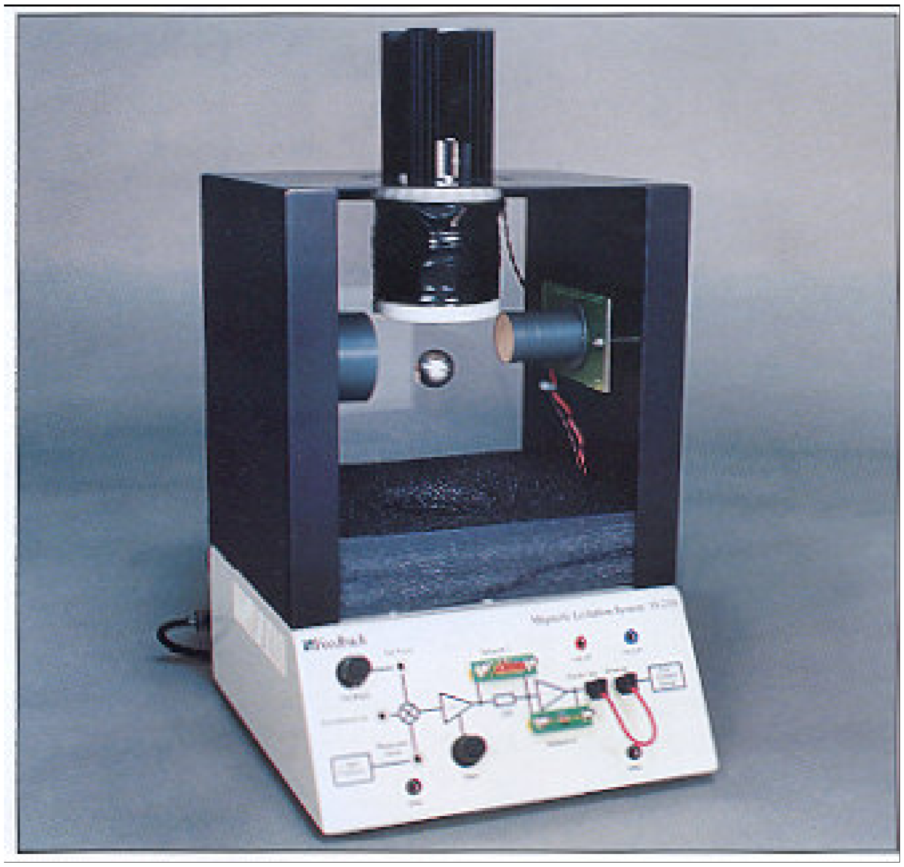


Figure 5-1: A photograph of the Magnetic Suspension System

A photograph of the Magnetic Suspension System is shown in Figure 5-1.

Block Diagram

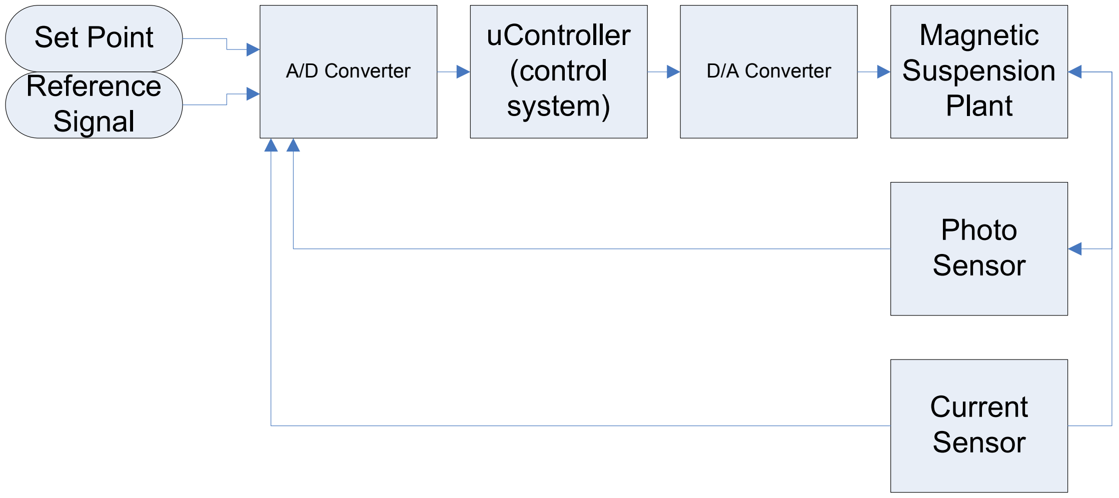


Figure 5-2: High Level System Block Diagram

Figure 5-2 shows a high level block diagram of the main components of the system: inputs, feedback signals, A/D converter, control system, D/A converter, plant, photo sensor, and current sensor.
The inputs are set point, a point at which the ball should be suspended and the reference signal or a signal which the ball should track about the set point. The feedback signals are current and position. A/D converter allows use of a digital controller, either the xPC or the ColdFire. The D/A converter will send the control signal to the plant converting from digital representation to analog voltage. The plant is the magnetic suspension system.

Modeling of Current Characteristics

a) Experimental
A frequency sweep must be performed on the Magnetic Suspension system to derive a plant model for the current feedback. Since the system can only be tested closed loop a summer circuit, shown in Figure 6-1, was built to inject a sinusoidal wave into the current driver to obtain a frequency response of just the voltage to current subsystem.

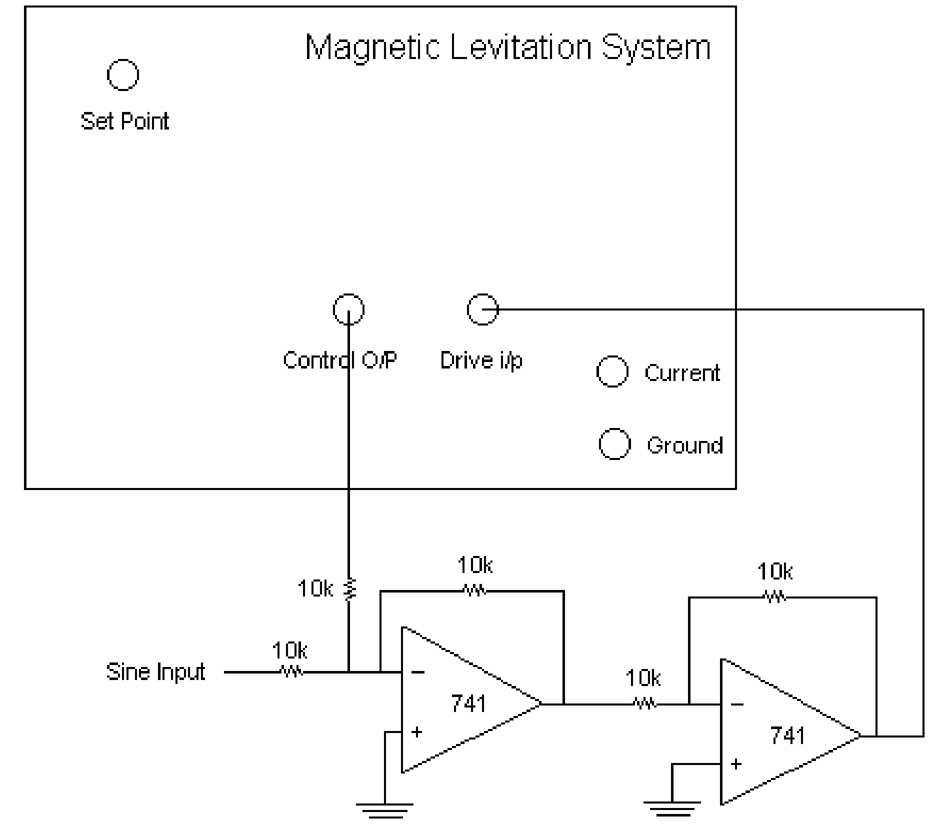


Figure 6-1: Analog Summer Circuit

From the frequency response of the Magnetic Suspension system a bode plot, shown in Figure 6-2, was plotted. A linear system would show normal characteristics of a 20dB or 40dB slope. The voltage to current transfer function had a low pass characteristic with +6dB/dec slope followed by a -15 dB/dec slope, which suggests nonlinear behavior.

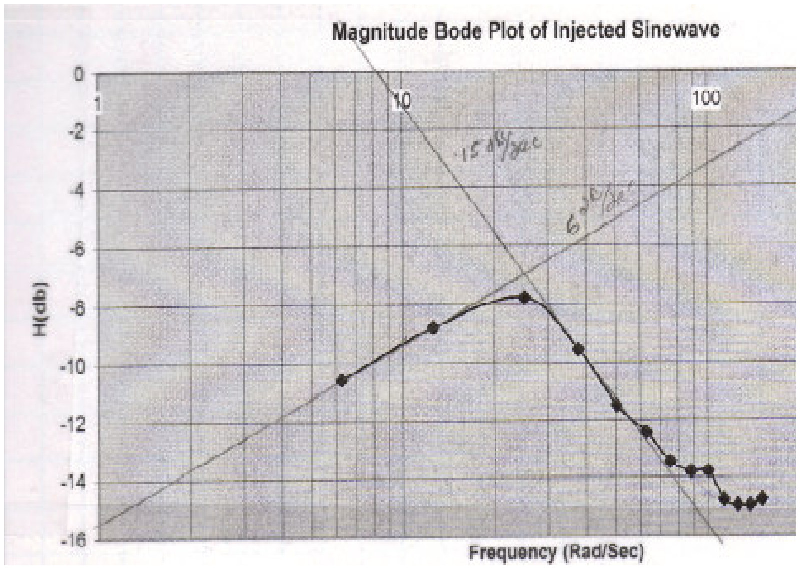


Figure 7-1: Experimental Frequency Sweep Plot

b) Simulink

Using data gathered experimentally an approximate model was developed in Simulink. Starting with a low pass filter and adding terms to try and fit a bode plot to experimental data, a 9th order model was created and its' bode plot is shown in figure 7-2. This model exhibited similar bode characteristics but proved useless in operation of simulating plant and controller operation.

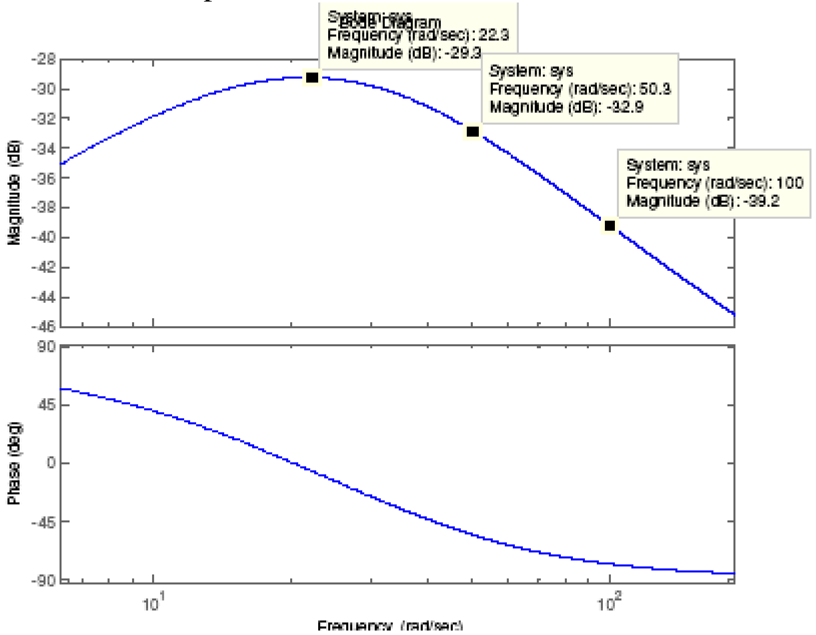


Figure 7-2: Simulation Frequency Sweep Plot

Since these results proved useless in simulation of actual plant and controller operation a second modeling approach was taken. A step response was captured on the plant measuring voltage and current traces. From this data a second nonlinear model was developed as shown in Figure 8-1. This model also demonstrated properties similar to the actual plant, but was not used in simulating actual operation.

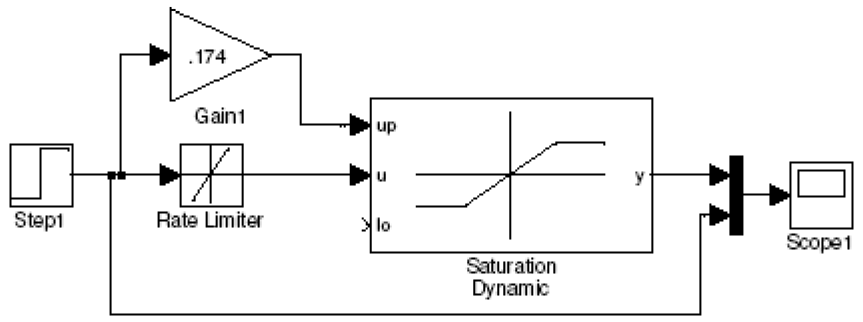


Figure 8-1: Nonlinear Model

The time domain response of the model in Figure 8-1 is shown in Figure 8-2. This matches closely the data taken from the plant with oscilloscope traces.

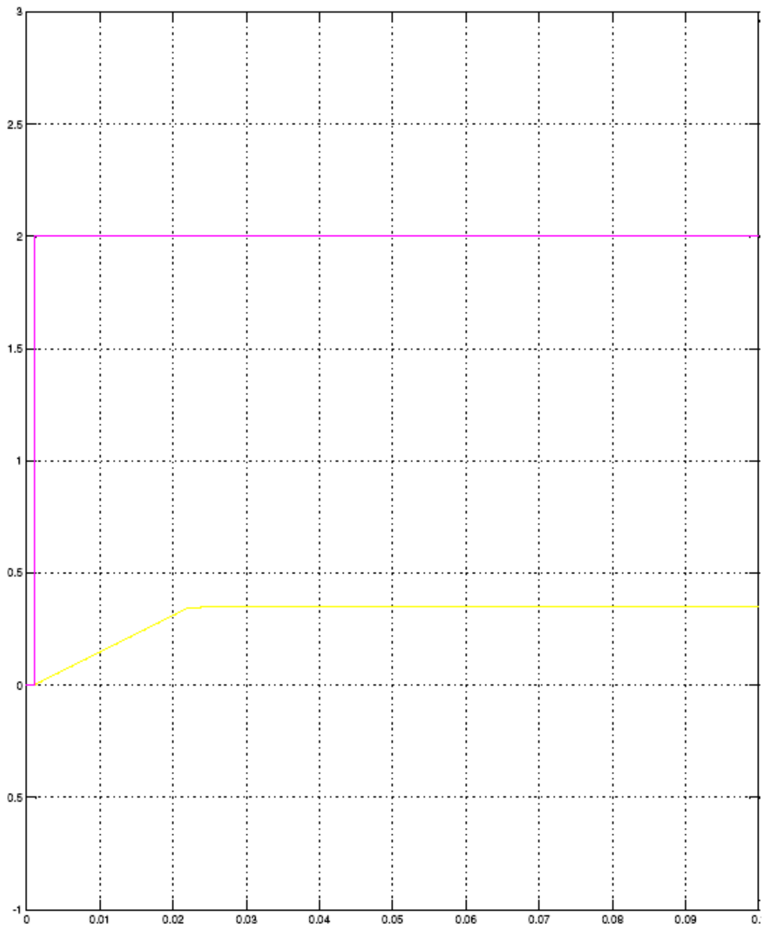


Figure 8-2: Nonlinear Model

Current Feedback Functionality

Using Simulink, a controller was developed to try and prove if current feedback, feasible in reality or not, would even help to produce a better controller. In an initial test, the model seen in Figure 9-1 was developed. This model was used to test initial condition response to steady state, seen in Figure 10-1. This initial test showed that current would in fact help, but later tests proved otherwise. The control approach was simple: create a current set point based on the position that the ball should be suspended at, and then subtract the actual current from that creating a current error signal which can be added to the output of the position loop controller to increase the performance of the plant.

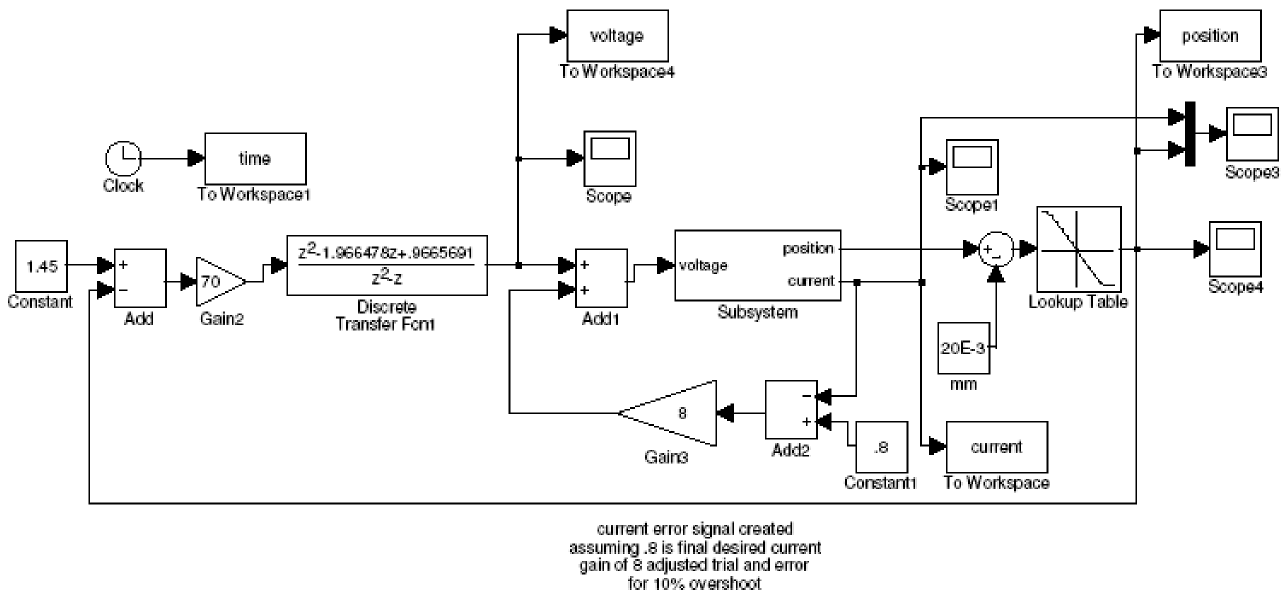


Figure 9-1: Initial Current Feedback Test Model

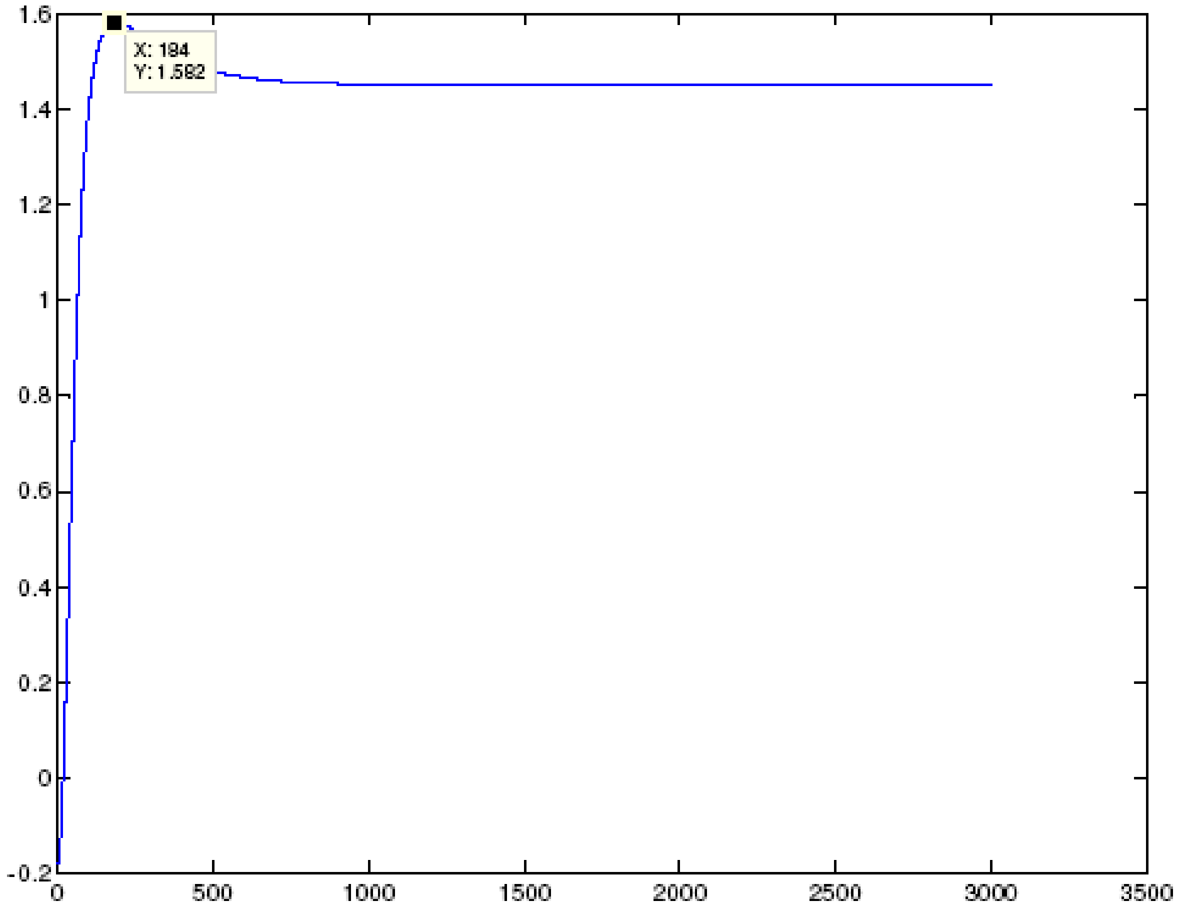


Figure 10-1: Initial Condition Response of Model

During later tests using controllers in simulation and on the xPC Targetbox, it was proven that any term added to the control signal via current feedback was integrated out by the position controller integrator. The position loop simply sees current feedback as disturbance. For this reason, in the current configuration, a controller using current feedback could not be developed.

Voltage, Current, and Position Relationship

The relationship between voltage and position was derived by previous students working on the same system. The goal of this project was to investigate a relationship between voltage, current, and position. Based on modeling attempts it appeared that a transfer function could not be derived linking voltage and current. Upon further investigation, it was discovered that current is not an independent variable as measured in this configuration because of the coil driver circuitry, see figure 4-1. The coil driver circuitry produces a current through the coil based on the voltage applied. The voltage is related to the position through a transfer function, but current is not independent of the voltage, therefore feeding back the current will not provide additional information about the state of the steel ball and developing a better controller utilizing current feedback is not possible.

Controller Design

Once it was proven that the addition of current feedback would not assist in the creation of a better controller the focus of the project was shifted to developing a better position feedback controller. This controller design was undertaken using two different methods. The first was an S-plane design, taken to the Z-plane for testing and implementation. The second was an all Z-plane design.

a) S-plane to Z-plane Design

In the first attempt at designing a better position only controller an inner-loop with outer-loop approach was adopted. For the inner loop a velocity controller would shift the unstable open loop poles closer to stability making for a simpler outer position loop controller. In Figure 11-1, the open loop unstable plant bode plot is seen in the top plot, and the inner loop less unstable bode plot is seen in the bottom plot. From this, an outer loop was developed to stabilize the system to specifications.

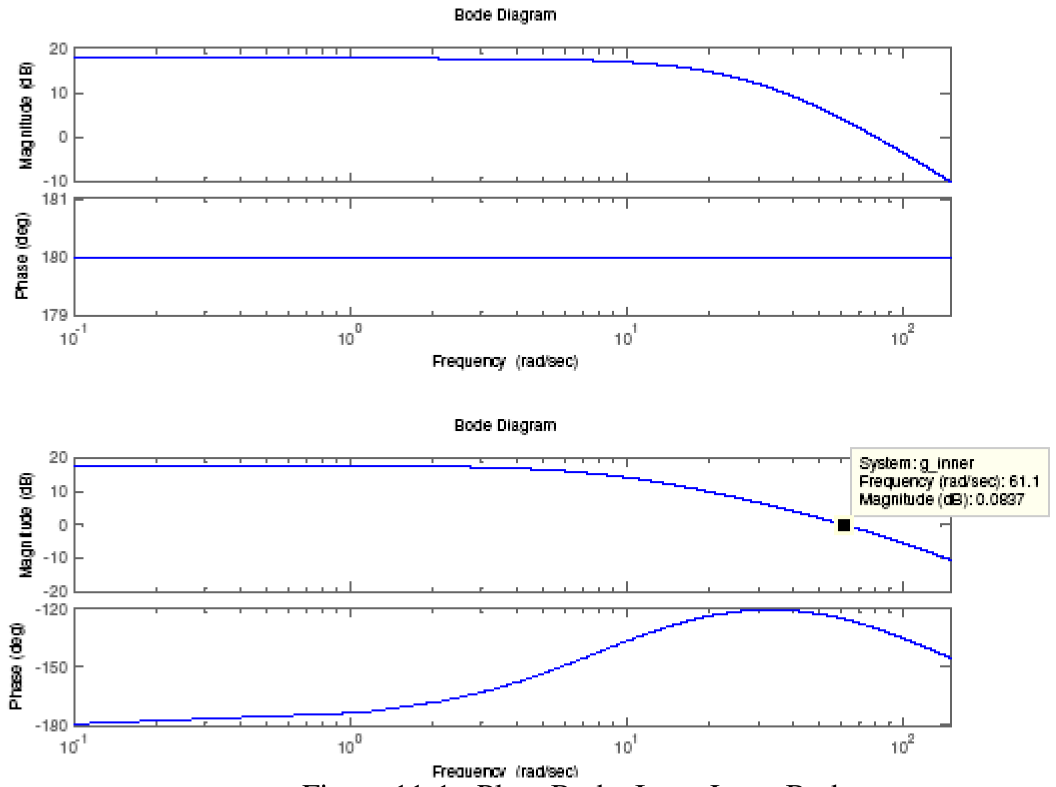


Figure 11-1: Plant Bode, Inner Loop Bode

The outer loop was designed using Root Locus technique. The root locus plot is shown in figure 12-1. We designed, according to the root locus plot, for a %OS of about 3%, which required a gain of about 0.767. In subsequent plots, the outer loop bode plot is shown, Figure 12-2, and finally the outer loop bode plot with desired gain compensation is seen in Figure 13-1.

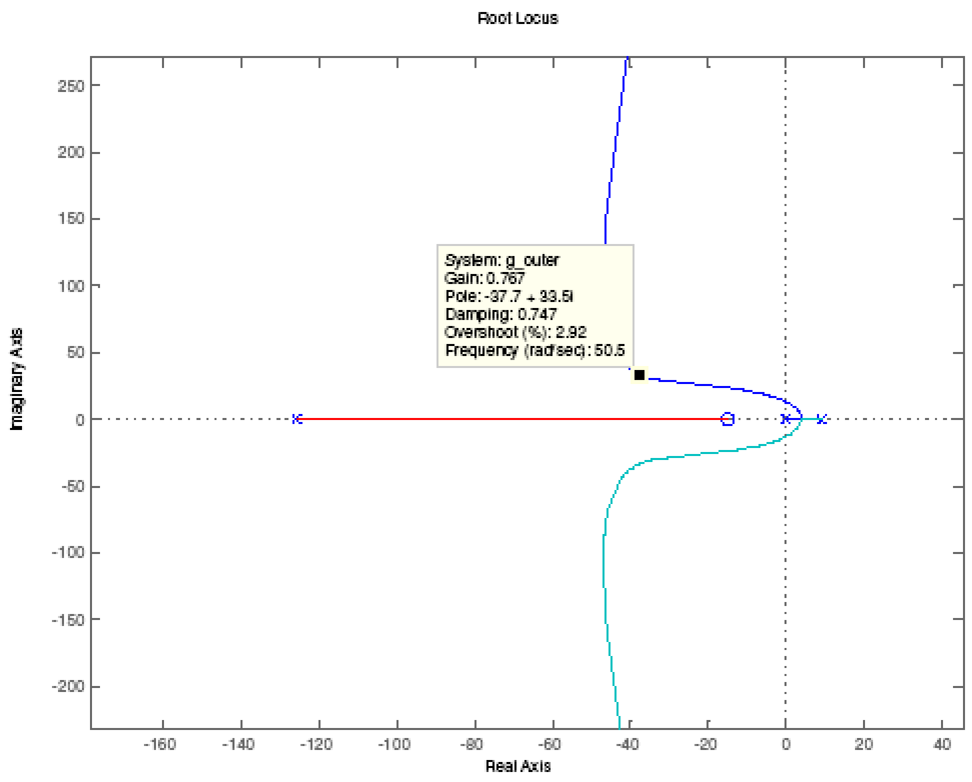


Figure 12-1: Outer Loop Root Locus Diagram

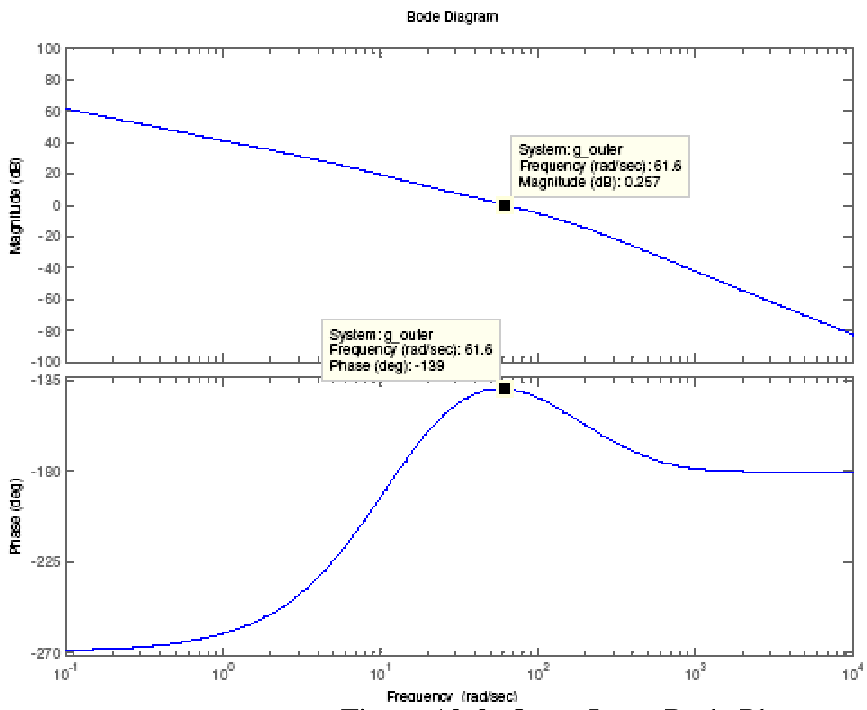
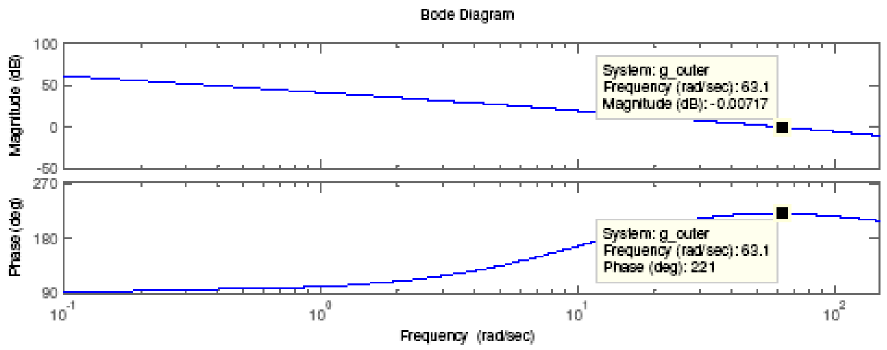


Figure 12-2: Outer Loop Bode Plot



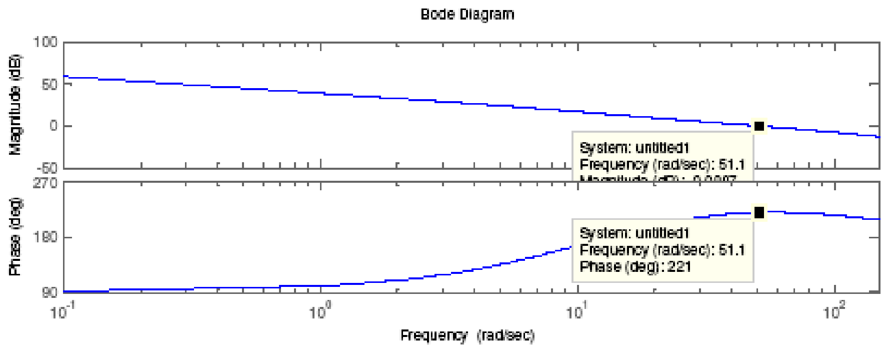


Figure 13-1: Outer Loop Bode Plot with Gain Compensation

This design worked perfectly in simulation. It produced slightly more overshoot than predicted by root locus, which is to be expected for a non-minimum phase system. However, the designed controller was never able to control the actual plant. The design was not stable enough to suspend the ball. This lead us to an all z-plane design approach.

b) All Z-plane Design

To do a completely digital design in the z-plane, the plant’s voltage to position transfer function was translated to the z-plane. From there, a controller was developed using root locus in the z-plane. Initially, an integrator was added to achieve e_ss=0, with its required pole, and a second pole was placed at 0.25, to bring the unstable open loop pole inside the unit circle. The zeros associated with and required by the desired pole placements where adjusted along with gain until a configuration was achieved that would stabilize the steel ball. The final root locus diagram is shown in Figure 14-1, along with an expanded view of the region around z=1 in Figure 14-2. The final controller design is seen in Figure 15-1, as mentioned above, poles where placed at 1, 0.25, and zeros were tuned to both be at 0.975 with an associated gain of 100. Step response in simulation is shown in Figure 15-2. M-file code used to arrive at these results can be found in Appendix A: MATLAB M-file Code.

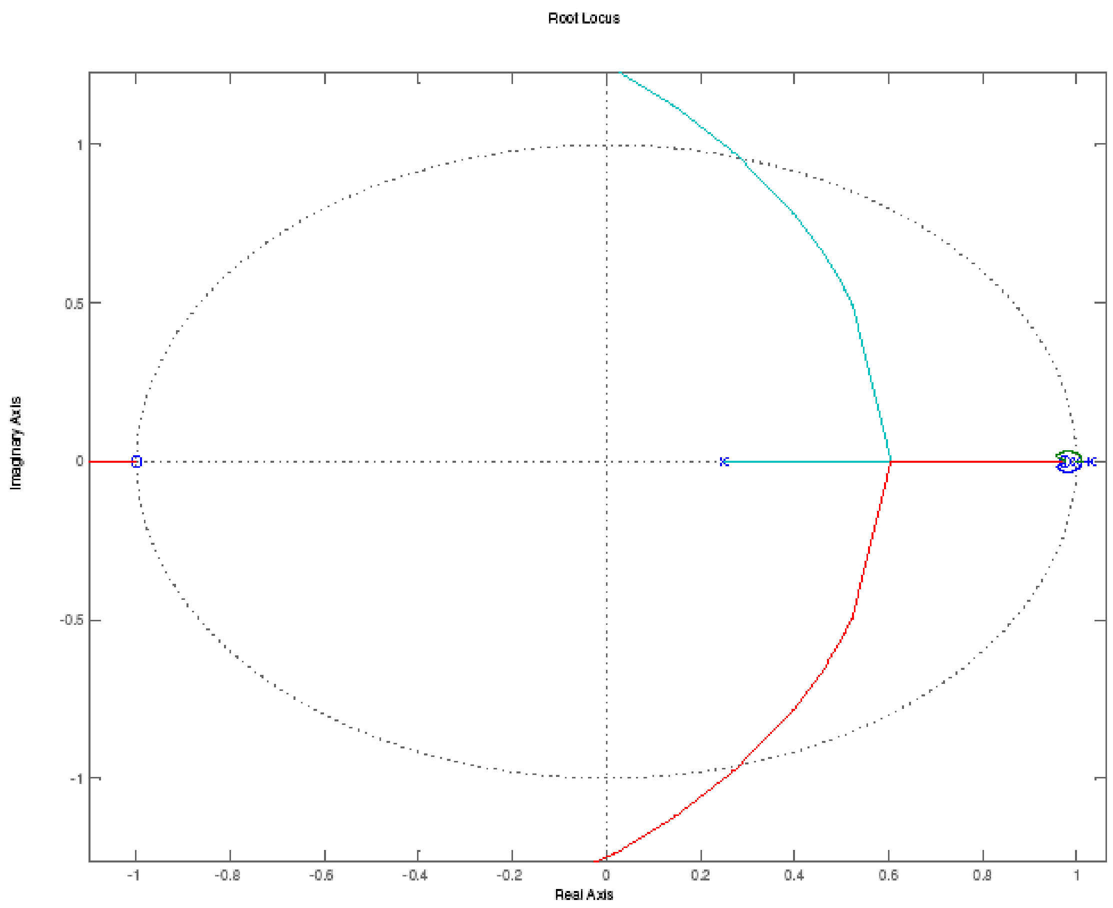


Figure 14-1: Z-Plane Root Locus

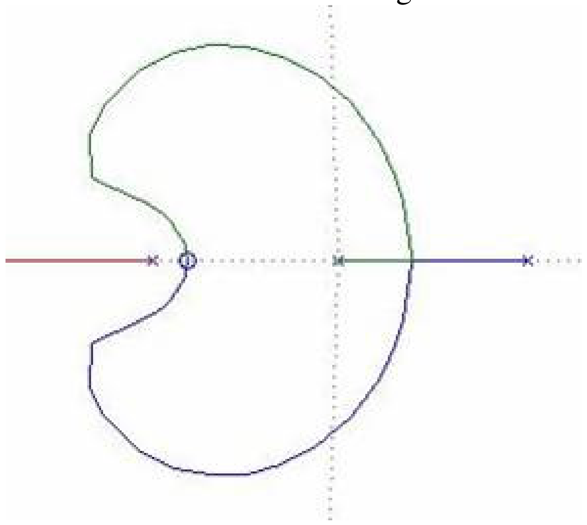


Figure 14-2: Expanded Root Locus near z=1

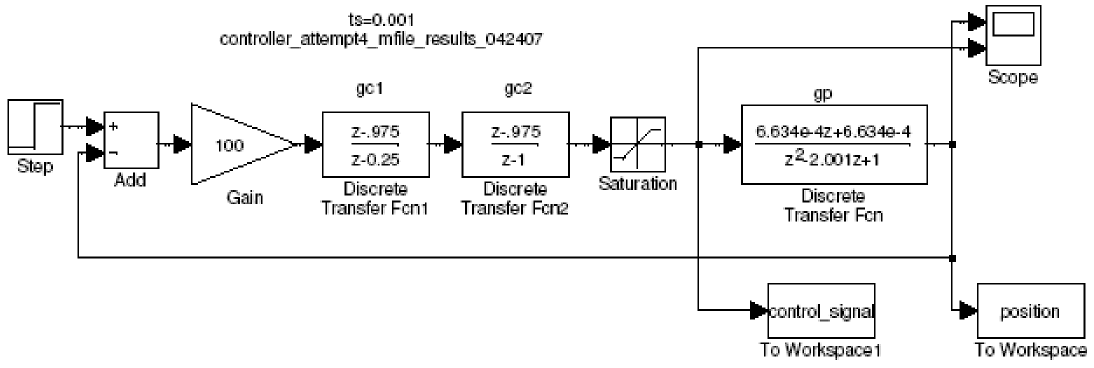


Figure 15-1: Digital Controller Design Model

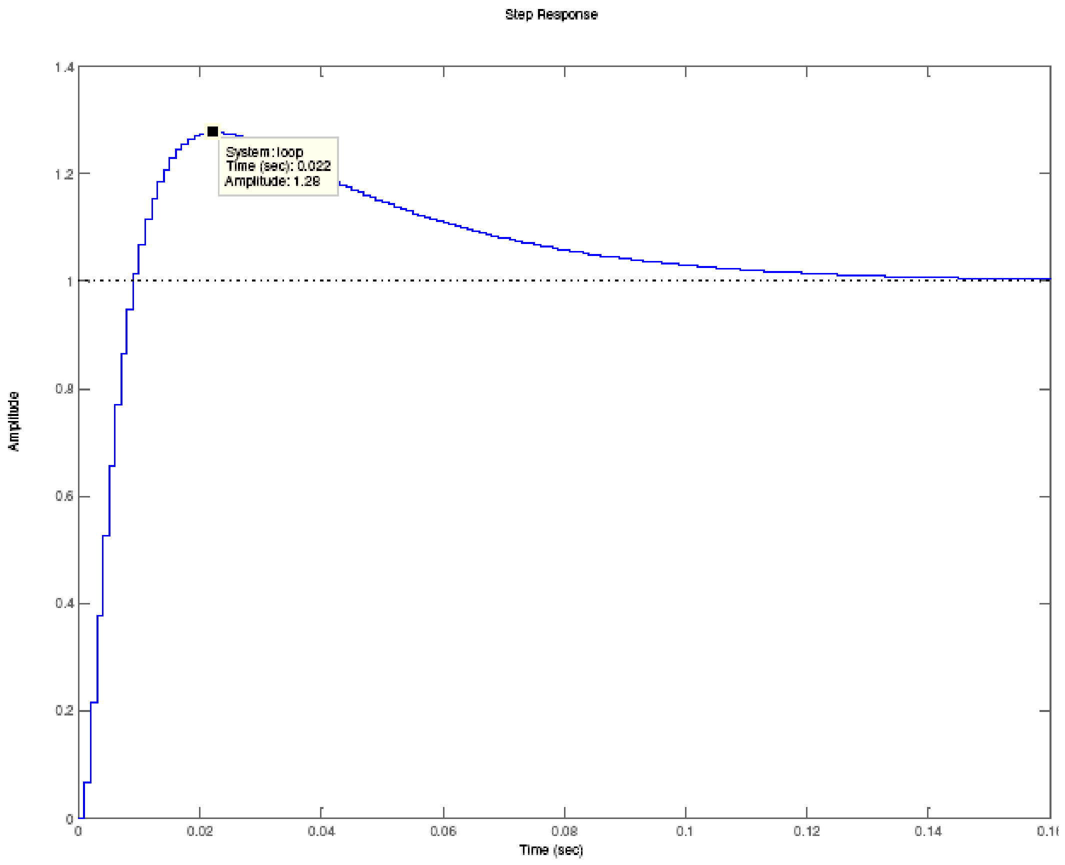


Figure 15-2: Digital Controller Step Response

xPC Implementation

In order to test the controller design, it was ported to the xPC Targetbox. A new model was created to handle the I/O and allow the digital controller to be executed on the xPC. This model is shown in Figure 16-1. Then using an I/O breakout header the signals were connected from the xPC to the plant.

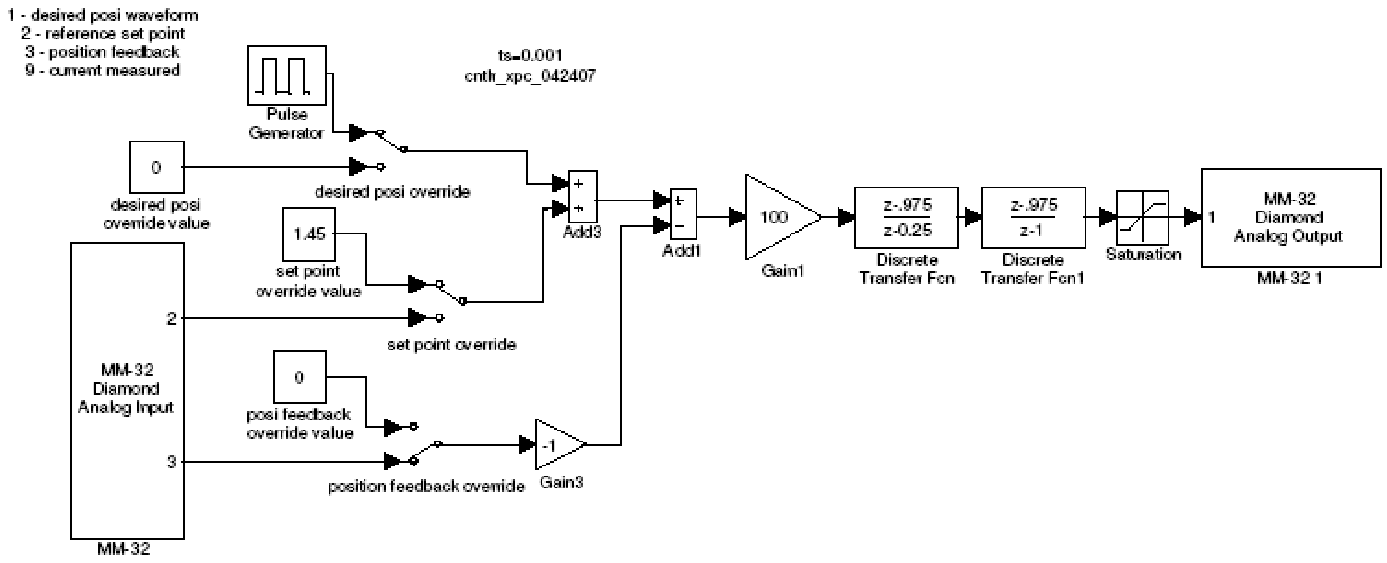


Figure 16-1: xPC Digital Controller

Microcontroller Implementation

a) Hardware

i) Overview

The Motorola ColdFire MCF5485 has a 32-bit processor, double precision floating point, DMA serial peripheral interface, and an interrupt controller. The MCF5485 is capable of performing at an operating frequency of 200 MHz or 308 MIPS. The development board used for this project did not have a built in Analog to Digital Converter (A/D Converter) or a Digital to Analog Converter (D/A Converter).

ii) Analog to Digital Converter

A MAX1202 was chosen as the A/D Converter as it is bipolar. The chip has 8 channels in which channel 0 was chosen for the input. The circuit diagram below shows the circuit connection of the converter. Since the A/D uses offset binary representation 0 volts in would read as 0x000h from the A/D. +2 volts is represented as 0x7FFh and -2 volts is represented as 0xFFFh.

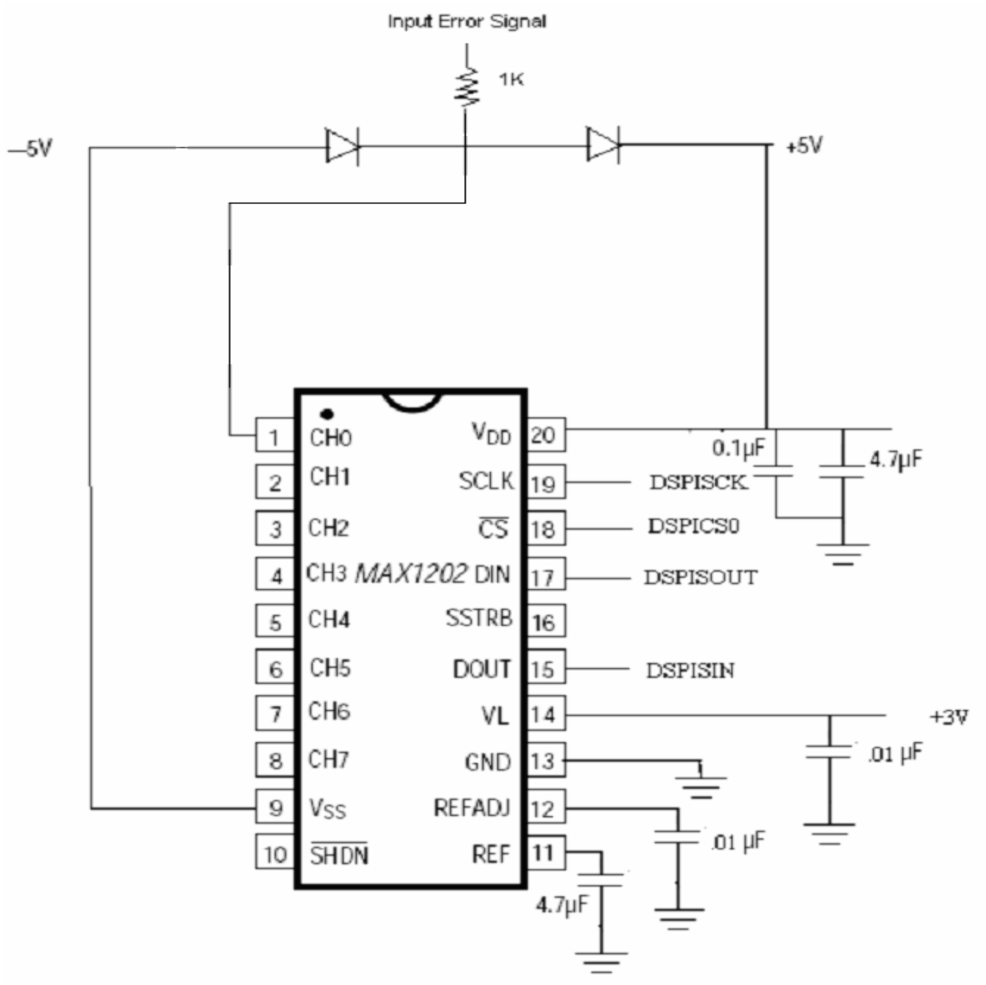
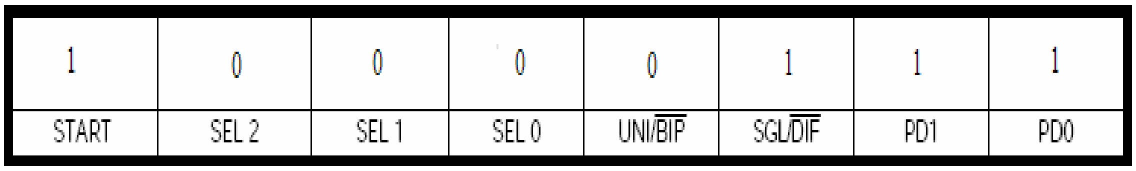


Figure 17-1: Analog to Digital Converter Circuitry

The pins involved in the data acquisition are SCLK, CS', DOUT, DIN. These communicated with the DSPI communication pins of the CML5485 MCU port. The clock ran at the rate of 1.04Mhz. Chip select pin0 was used for selecting the chip. The DIN pin was connected to DSPICOUT, while the DOUT was connected to the DSPISIN.

The processor operates as the master sending signals to the peripherals. The ADC then needs the control word from the processor whenever it is ready to get the data. The eight bit control word is formed as 87h.



1	0	0	0	0	1	1	1
START	SEL 2	SEL 1	SEL 0	UNI/$\overline{\text{BIP}}$	SGL/$\overline{\text{DIF}}$	PD1	PD0

Figure 17-2: Analog to Digital Control Register

In the code, 16 bits of data are sent first with the LSB being the control word and then MSB being zeros. In the next cycle another 16 bits are fetched from the converter, in which 12 would be the data and the rest are zeros.

iii) Digital to Analog Converter

A MAX5253 was chosen as the D/A converter because it has 4 channels with an output resolution of 12 bit. The circuit diagram below shows the circuitry attached to the D/A Converter. The R1 and R2 resistors were 10.1kohm and 9.9kohm. The reference signal fed was chosen as 2.0V to obtain the range +/-2V at the output. Since the D/A is in 2's complement, when the D/A outputs 0x000h the control signal should read -2V. An output of 0x800h from the D/A should correspond to 0V as the control signal. Finally, at 0xFFFh the control signal should read +2V.

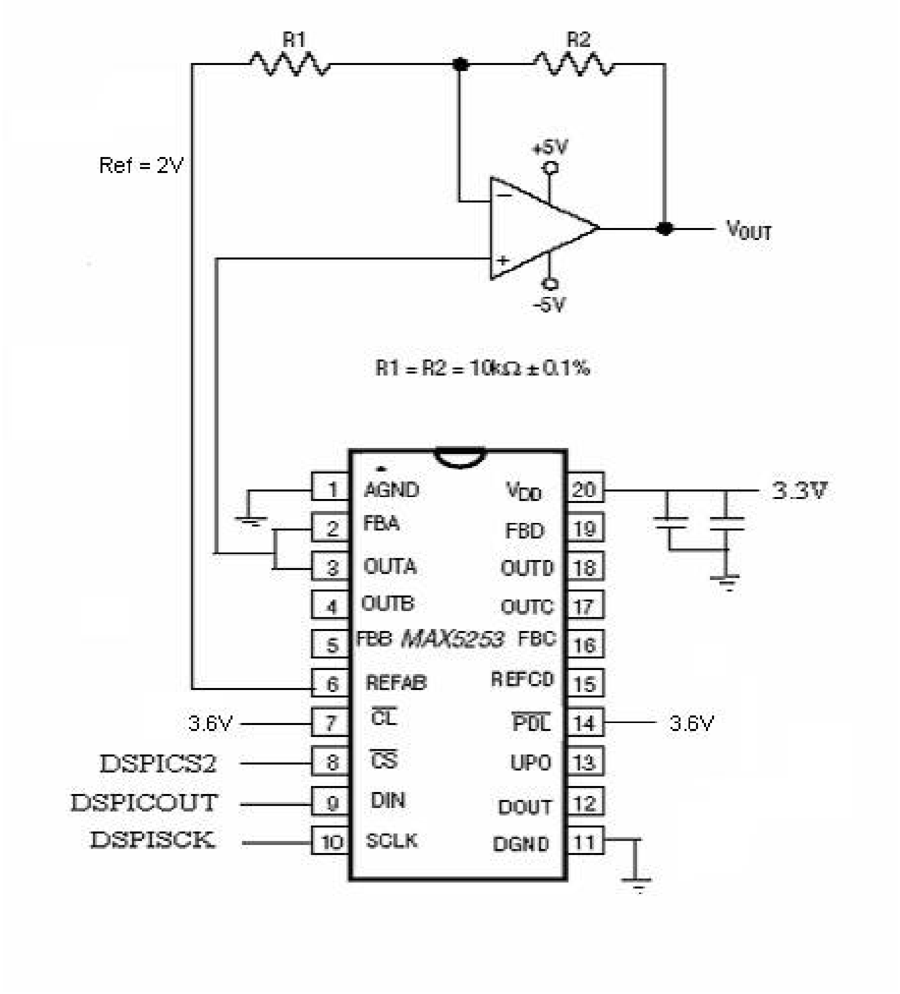


Figure 18-1: Digital to Analog Converter Circuitry

The pins involved in the data acquisition are SCLK, CS', DIN . These communicated with the DSPI communication pins of the CML5485 MCU port. The clock ran at a rate of 1.04Mhz. Chip select pin2 was used for selecting the chip. The DIN pin was connected to DSPICOUT while, the DOUT can be used for checking the data.

b) Difference Equation

In the real-world control applications all systems are nonlinear to some extent, and have parameters that vary with time to some degree. The majority of systems can be mathematically modeled with linear time invariant differential equations. Once this is completed the effects of sampling the input and updating the output at discrete intervals can be taken into account and a difference equation that describes the system behavior can be implemented using any software. Below are the steps used to obtain our difference equation.

I) Controller Transfer Function (Z-plane)

$$Gc(z) = \frac{z^2 - 1.95z + 0.950625}{z^2 - 1.25z + 0.25}$$

II) Divide by Highest Z power

$$\frac{U(z)}{E(z)} = \frac{1 - 1.95z^{-1} + 0.950625z^{-2}}{1 - 1.25z^{-1} + 0.25z^{-2}}$$

III) Cross Mulitply

$$U(z) - 1.25z^{-1} \cdot U(z) + 0.25z^{-2} \cdot U(z) = E(z) - 1.95z^{-1} \cdot E(z) + 0.950625z^{-2} \cdot E(z)$$

IV) Inverse Z-Transform

$$u(n) = 1.25u(n-1) - 0.25u(n-2) + e(n) - 1.95e(n-1) + 0.950625e(n-2)$$

From the result in step IV, C++ code is written to implement the equation

c) Software Flowchart

The overall process used to implement the ColdFire microcontroller can be seen in the software flowchart in Figure 20-1. The Timer is first initialized and then there is an infinite loop that repeats every 1ms to read in the error signal off the A/D and then run through the software controller and output to the control signal on the D/A.

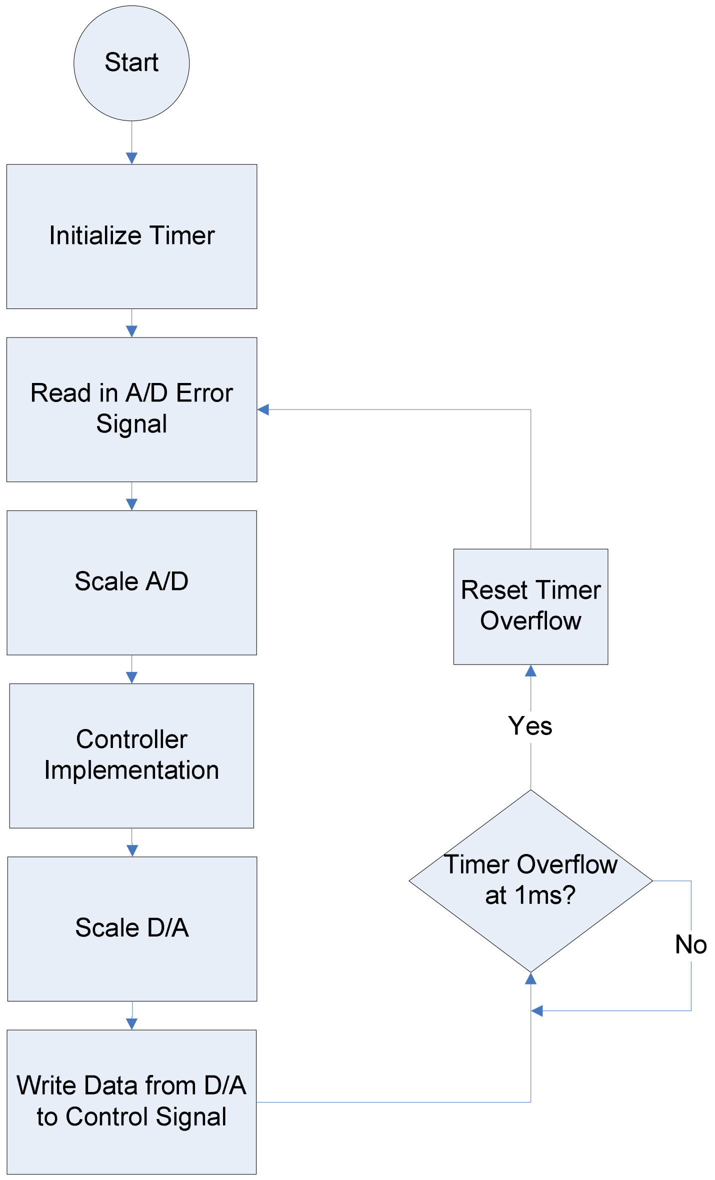


Figure 20-1: Software Flowchart

d) Software Implementation

i) Timer

Most of the following code was written by Shobana Mahadevan, a former graduate student at Bradley University. Additions to the code have been made to implement a fixed sample time of 1ms and to implement a new controller based on the difference equation obtained from the previous page. For an in depth explanations of the code please refer to Shobana Mahadevan's final report titled, "Control of magnetic levitation system using Coldfire Processor MCF5485". The complete code can be found in Appendix 1.

The code directly below is used to initialize a timer for 1ms to create a fixed sampling period. After the overflow flag is set, it must be manually reset. On the MCU Bus of the Microcontroller pin 18 is complemented to show how long just the code takes to run. Pin 20 is complemented to show the fixed sampling period of 1ms. See Figure 1 of pin outputs.

```
MCF_GPT_GSR0 = MCF_GPT_GSR_TEXP; //clears timer overflow flag
        //configures Timer for 1ms timing with prescalar of 4
MCF_GPT_GCIR0 = 0x02c350;
MCF_GPT_GMS0 = 0x1007;  // for internal timer
MCF_GPIO_PDDR_PSC3PSC2  = 0x11; //sets pins 18 and 20 as output pins

while(1) { // start complete loop code

while ((MCF_GPT_GSR0 & 0x8)==0) //wait for 1 ms (for fixed period)
{}

MCF_GPT_GSR0 = MCF_GPT_GSR_TEXP; //clears timer overflow flag
MCF_GPT_GMS0 = 0x0;
MCF_GPT_GCIR0 = 0x2c350;  //configures Timer for 1ms timing with
MCF_GPT_GMS0 = 0x1007;   // prescalar of 4 for internal timer

//(pin 18) used to see controller execution time
MCF_GPIO_PODR_PSC3PSC2 = MCF_GPIO_PODR_PSC3PSC2 ^ 0x10;

//(pin 20) used to see timer period
MCF_GPIO_PODR_PSC3PSC2 = MCF_GPIO_PODR_PSC3PSC2 ^ 0x01;
…
} //end complete loop code
```

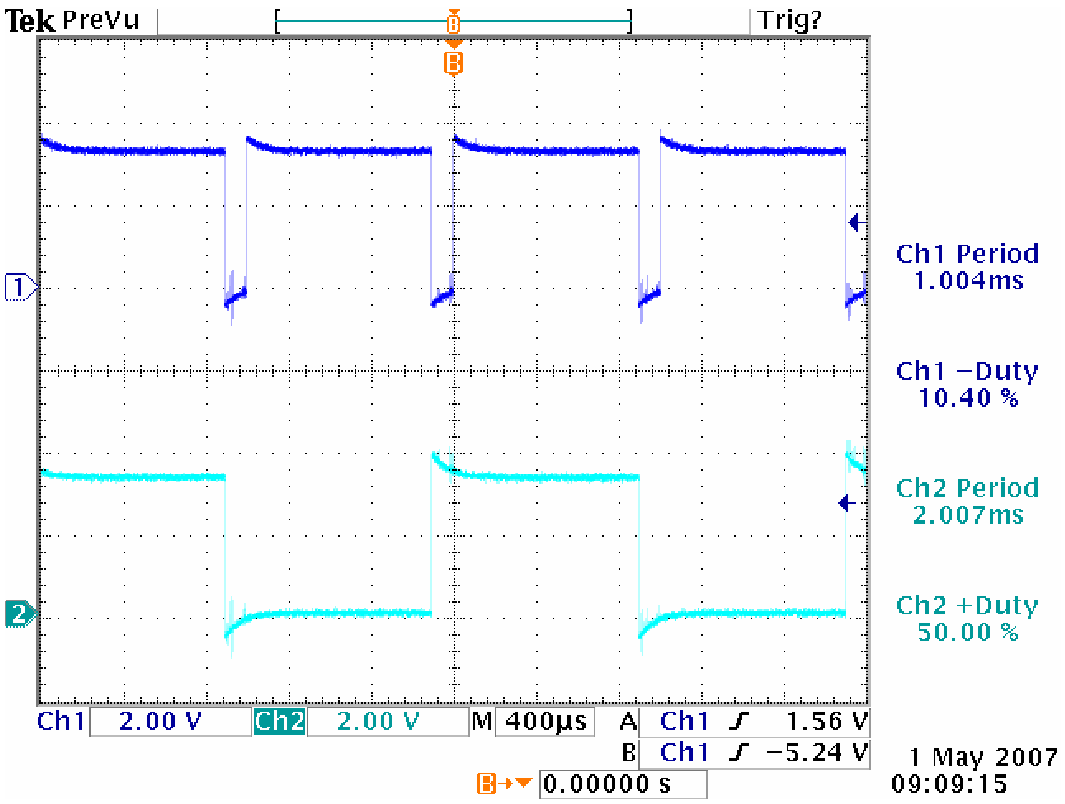


Figure 22-1. Timing on pins 18 and 20 respectively

The General Purpose Timer (GPT) on the ColdFire Processor was used to create the 1ms timing. There are four registers that are associated with the GPT. They are Enable and Mode Select Register, Counter Input Register, PWM Configuration Register and Status Register. In the Enable and Mode Select Register only CE and TMS need to be all ones. The rest of the register should be set to zeroes. The timer will run in stop mode. A picture of the Enable and Mode Select Register can be seen below in Figure 22-2.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OCPW								0	0	OCT		0	0	ICT	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WDEN	0	0	CE	0	SC	OD	IEN	0	0	GPIO		0	TMS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reg Addr	MBAR + 0x800 (GMS0), 0x810 (GMS1), 0x820 (GMS2), 0x830 (GSM3)															

Figure 22-2. GMS Register

The Counter Input Register is 32 bits long with the upper word as a prescaler and the bottom word as the count. Since the processor operates at 200 MHz there is a prescaler value of 4 with a counter of 0xC350h. This will then generate an overflow at 1ms.

The last register used is the Status Register. The Status Register holds the timer overflow flag TEXP that is used to check when the 1ms is up. The following line of code is used to reset the overflow flag.

```
MCF_GPT_GSR0 = MCF_GPT_GSR_TEXP;
```

ii) Controller
After creating a fixed sampling period of 1ms it was time to implement the difference equation, found on the previous page, in software. The following code scheme is used to obtain the software controller. The error signal that comes into the A/D is put in rdata2.Microcontroller Implementation: Software Implementation

```
en=rdata2;                    //data received from the A/D Converter
value1  = 0.001*en;    //Scaling(1bit = 1mV)
value2  = 0.001*enm1;
value3  = 0.001*enm2;

// note 75= overall controller gain compensation
value1 = (75)*value1; //Difference equation implementation
value2 = (75)*(1.95)*value2;
value3 = (75)*(0.950625)*value3;
Tvalue = value1-value2;
Tvalue = Tvalue+value3;
Tvalue = Tvalue+(1.25*unm1);
Tvalue = Tvalue-(0.25*unm2);
un = Tvalue;

Tvalue=Tvalue/.001; // scalar for DAC

data2dac =Tvalue;
data2dac=data2dac^0x800;
data2dac=data2dac&0xFFF;

enm2=enm1;
enm1=en;
unm2=unm1;
unm1=un;
```

e) Results

To test the response of the new position controller implemented in software a step response with an amplitude of 0.2Vpp at 0.5Hz was applied to the suspended ball. The picture of the step response can be seen below in Figure 24-1.

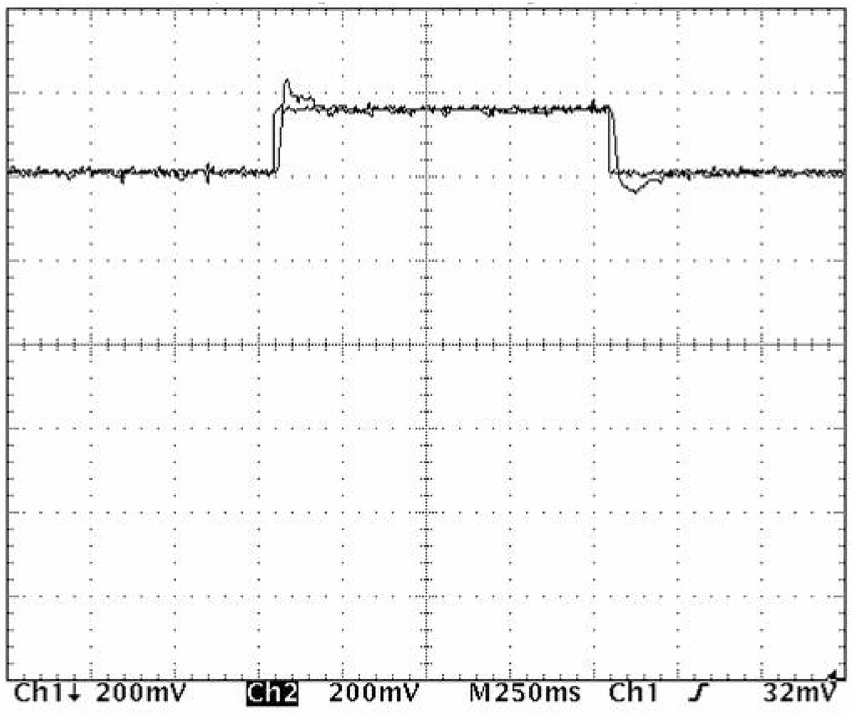


Figure 24-1. Step Response of Microcontroller

Using an oscilloscope the overshoot was measured at 46% and the settling time was 240ms. To test out the range of operation of just the suspended ball without any tracking the set point was slowly increased until the ball was pulled into the electromagnet. The reverse was also performed to see the lower bound of operation. In all, the ball could remain suspended if the set point was in a range from +2.8V to -0.9V.

Another test was also performed to see how long the controller implemented on the ColdFire processor could stabilize the ball tracking a sinusoidal wave at 1Vpp amplitude. The ball remained stable until 5 Hz, at which point the ball dropped.

f) Operating Procedure

Follow these steps to run the controller on the ColdFire:

1.) Turn on all power supplies. (+5V, -5V, +2V, +3.6V)
2.) Plug in power to microcontroller
3.) Attach A/D input to error signal on Magnetic Suspension System
4.) Attach D/A output from op amp to control signal on Magnetic Suspension System
5.) Log into computer next to the microcontroller
6.) On the desktop there is a folder named “Coldfire Code” with all of the code

7.) Open CodeWarrior and select “open project”. The project is located in the subdirectory of the *Coldfire Code* folder on the desktop
8.) Once the project is loaded, open *console_main.c*. This is the file with all of the code.
9.) Now turn on the Magnetic Suspension system and place the steel ball on the holder and place it under the electromagnet.
10.) Using an oscilloscope measure the error signal and adjust the ball height until the error signal is around 200mV or less.
11.) Go to code warrior and click run.
12.) The ball should be suspended. Lower the ball holder to verify.

Recommendation for Future Work

There are several different paths that can be taken for continual project work. If one were to design the hardware of the Magnetic Suspension System from the bottom up then current feedback could become a viable controller design. Currently, there are frequencies well above our crossover frequency that are affecting the overall performance and stability of the system. At this time it is unclear what the effects of adding an anti-aliasing filter would do the system, but it is believed that adding this filter would remove all unwanted frequencies, thus improving stability and performance. The microcontroller code written is setup perfectly to communicate with the A/D and D/A converters. Organization of the code was not taken into consideration and some time could be spent modularizing it. Once a route has been chosen for a design, realistic specifications should be included.

Conclusion

The goal of this project was to determine if current feedback could be utilized to improve performance of the Magnet Suspension System. Through current characteristic modeling and controller implementation, in Simulink and experimentally, it was determined that current feedback could not be used. The focus of the project was then shifted to developing an improved performance position controller. A new controller was designed using root locus techniques, tested in Simulink simulation, and implemented on the xPC targetbox and the Coldfire microcontroller. The controller exhibited less stability and more overshoot, but showed a dramatic reduction in settling time.

References

Jose A Lopez and Winfred K. N. Anakwa, “Identification and Control of a Magnetic Suspension System using Simulink and dSPACE Tools”, Proceedings of the ASEE Illinois/Indiana 2003 Sectional Conference, March 27, 2004, Peoria, Illinois, U.S.A.

Feedback Inc., 437 Dimmocks Mill Road, Hillsborough, North Carolina 27278. http://www.fbk.com

Simulink, Version 6.1 (R14SP1), The MathWorks Inc., Natick, MA 01760, 2004.

xPC Target Box, The MathWorks Inc., Natick, MA 01760, 2003.

Motorola ColdFire MCF5485RM, Rev. 4, Freescale Semiconductor, Inc. 2006.

Appendices

Appendix a: MATLAB M-File Code

M-File code to design new position controller in Z-plane via Root Locus

```
clc
ts=0.001;
gp_n=[0 .18*7.67*961];
gp_d=[1 0 -961 ];
%take plant model to z-plane
[gpnum,gpden]=c2dm(gp_n,gp_d,ts,'zoh');
disp('gp')
%gp_sys=tf(gpn,gpd,ts)
gp_sys=tf(gpnum,gpden,ts)
%
% gp
%
% Transfer function:
% 0.0006634 z + 0.0006634
% -----------------------
%    z^2 - 2.001 z + 1
%
%Sampling time: 0.001
%
%subplot(2,1,1),rlocus(gp_sys)
gc1n=[1 -0.975];
gc1d=[1 -0.25];
gc2n=[1 -0.975];
gc2d=[1 -1];
gc1_sys=tf(gc1n,gc1d,ts);
gc2_sys=tf(gc2n,gc2d,ts);
disp('gc')
gc_sys=series(gc1_sys,gc2_sys)
%
% gc
%
% Transfer function:
% z^2 - 1.95 z + 0.9506
% ---------------------
%  z^2 - 1.25 z + 0.25
%
% Sampling time: 0.001
%
%subplot(2,1,2),rlocus(gc_sys)
disp('g_all')
g_all=series(gc_sys,gp_sys)
%
% g_all
%
% Transfer function:
% 0.0006634 z^3 - 0.0006303 z^2 - 0.000663 z + 0.0006307
% ------------------------------------------------------
%      z^4 - 3.251 z^3 + 3.751 z^2 - 1.75 z + 0.25
%
% Sampling time: 0.001
%
```

```
%g_all=tf(g_alln,g_alld,ts)
rlocus(g_all)
%zgrid
%k=[100 200 300 400];
r=rlocus(g_all,100) %300 looks like about 5% overshoot
disp('loop')
loop=feedback(100*g_all,1.0,-1.0)
%
% loop
%
% Transfer function:
% 0.06634 z^3 - 0.06303 z^2 - 0.0663 z + 0.06307
% ----------------------------------------------
% z^4 - 3.185 z^3 + 3.688 z^2 - 1.817 z + 0.3131
%
% Sampling time: 0.001
%
%step(loop)
```

Appendix b: Cold-Warrior C++ Code

Codewarrior C++ code to implement adc, dac, and difference equation.

```
#include "src/include/common.h"
#include "src/include/io.h"

        volatile unsigned short flag;

void main (void)
{

        signed short int rdata2,rdata3,data2dac;
        signed long int rdatDac,rdata1,c1,data;
        double Tvalue, value1, value2, value3, value4, value5, un, unm1, unm2,
        en, enm1, enm2;
        int i=0; en=0;enm1=0;enm2=0;un=0;unm1=0;unm2=0;


        MCF_GPT_GSR0 = MCF_GPT_GSR_TEXP; //clears timer overflow flag
        //configures Timer for 1ms timing with prescalar of 4
        MCF_GPT_GCIR0 = 0x02c350;
        MCF_GPT_GMS0 = 0x1007;  // for internal timer
        MCF_GPIO_PDDR_PSC3PSC2  = 0x11; //sets pins 18 and 20 as output pins

while(1)
{

while ((MCF_GPT_GSR0 & 0x8)==0) //wait for 1 ms (for fixed period)
{}

        MCF_GPT_GSR0 = MCF_GPT_GSR_TEXP;  //clears timer overflow flag
        MCF_GPT_GMS0 = 0x0;
        MCF_GPT_GCIR0 = 0x2c350;  //configures Timer for 1ms timing with
        MCF_GPT_GMS0 = 0x1007;   // prescalar of 4 for internal timer

        //(pin 18) used to see controller execution time
        MCF_GPIO_PODR_PSC3PSC2 = MCF_GPIO_PODR_PSC3PSC2 ^ 0x10;

        //(pin 20) used to see timer period
        MCF_GPIO_PODR_PSC3PSC2 = MCF_GPIO_PODR_PSC3PSC2 ^ 0x01;

        MCF_GPIO_PDDR_DSPI = 0x60;

        //Toggling of the bit
        MCF_GPIO_PODR_DSPI = 0x40;
MCF_GPIO_PODR_DSPI = 0x00;

//initialization of the DSPI to communicate with ADC
        MCF_GPIO_PAR_DSPI       = 0x00FF;
        MCF_DSPI_DMCR = 0x81010000;

//configures for 8bit transfer and the clock rate

        MCF_DSPI_DCTAR0     = 0x78553335;

        //configures for 16bit transfer and the clock rate
        MCF_DSPI_DCTAR1 = 0x78553335;
        MCF_DSPI_DSR = MCF_DSPI_DSR_TCF;
        MCF_GPIO_PODR_DSPI = 0x40;

        //select the attribute reg and asserts the CS
        MCF_DSPI_DTFR = 0x04010087;

        while(!(MCF_DSPI_DSR & MCF_DSPI_DSR_TCF)){};
                MCF_GPIO_PODR_DSPI = 0x00;
                MCF_DSPI_DSR=0x90000000;
                rdata1 = MCF_DSPI_DRFR;

                MCF_GPIO_PODR_DSPI = 0x40;
                MCF_DSPI_DTFR = 0x14010000;

        while(!(MCF_DSPI_DSR & MCF_DSPI_DSR_TCF)){};
                MCF_GPIO_PODR_DSPI = 0x00;
                MCF_DSPI_DSR=0x90000000;
                rdata2 = MCF_DSPI_DRFR;

        rdata2=rdata2<<1;
        rdata2=rdata2/16;

// Controller
        en=rdata2;
        value1  = 0.001*en;
        value2  = 0.001*enm1;
        value3  = 0.001*enm2;

        //Gary's
        value1 = (75)*value1;
        value2 = (75)*(1.95)*value2;
        value3 = (75)*(0.950625)*value3;
        Tvalue = value1-value2;
        Tvalue = Tvalue+value3;
```

```
            Tvalue = Tvalue+(1.25*unm1);
            Tvalue = Tvalue-(0.25*unm2);
            un = Tvalue;

            Tvalue=Tvalue/.001; // scalar for DAC

            data2dac =Tvalue;
            data2dac=data2dac^0x800;
            data2dac=data2dac&0xFFF;

            enm2=enm1;
            enm1=en;
            unm2=unm1;
            unm1=un;

            rdatDac=data2dac;//data for the DAC

        //  config for DAC
            MCF_GPIO_PAR_DSPI       = 0x033F;
            MCF_DSPI_DMCR = 0x81040000;
            MCF_DSPI_DCTAR0 = 0x78553335;
            MCF_GPIO_PODR_DSPI = 0x20;

            c1=(rdatDac&0xFFF)|0x00043000;

            MCF_DSPI_DTFR = c1;//c10x10043FFF;

        while(!(MCF_DSPI_DSR & MCF_DSPI_DSR_TCF)){};
            MCF_GPIO_PODR_DSPI = 0x00;
            MCF_DSPI_DSR = 0x90000000;

            rdatDac = MCF_DSPI_DRFR;

        MCF_GPIO_PODR_PSC3PSC2 = MCF_GPIO_PODR_PSC3PSC2 ^ 0x10;

    }
    }
```