Design of Magnetic Levitation Controllers Using Jacobi Linearization, Feedback Linearization and Sliding Mode Control

Bharathwaj ”Bart” Muthuswamy and Kevin Peterson.

I. Introduction (BART AND KEVIN)

THE goal of this project was to design three (one linear;two nonlinear) magnetic levitation controllers for the system shown in Fig. 1. Despite the fact that magnetic levitation systems are described by nonlinear differential equations [2], a simple approach would be to design a controller based on the linearized model (Jacobi linearization about a nominal operating point [6]). But this means the tracking performance deteriorates rapidly with increasing deviations from the nominal operating point. Nevertheless, a controller based on Jacobi linearization is a good ”litmus test” for our system identification, hence this controller is designed first.

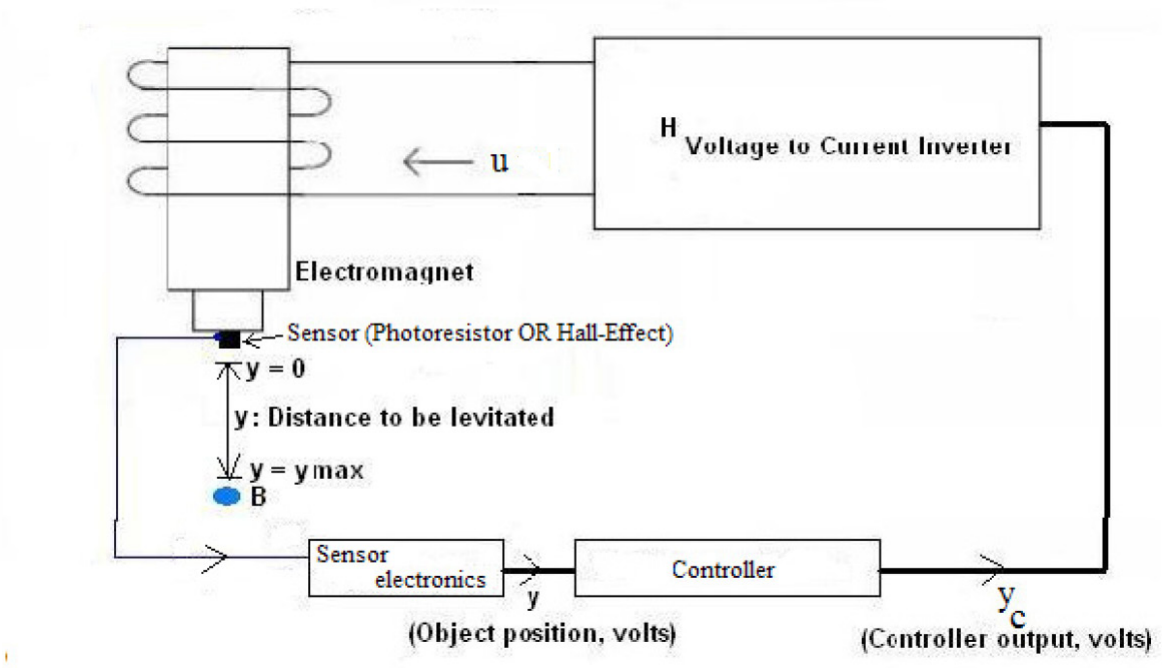

Fig. 1: A block diagram of the magnetic levitation system and the feedback control system.

The objectives of our control strategy are:

1) **Linear Controller:** Levitate a metallic sphere (ball) over a distance of approximately 0.1 cms [6].
2) **Nonlinear Controller(s):** Levitate a hex nut over a distance of approximately 1.6 cms. These controllers should also be able to track specified input (sinusoidal, sawtooth etc.) trajectories. The

Bart and Kevin are with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, 94720 USA e-mail: mbharat@cory.eecs.berkeley.edu, kevincp@eecs.berkeley.edu

design strategies for these controllers form the crux of this report. The principal references for the nonlinear control design are [9] and [2].

An outline of the two proposed nonlinear control strategies are:

1) **Feedback Linearization (Kevin and Bart)**.Through the use of feedback linearization, we can transform the system dynamics from complicated nonlinear ones to more simple linear dynamics. The linear model can now be controlled through the use of linear state feedback. This method gives us an advantage over Jacobian linearization by allowing the control of a simple linear plant without neglecting the nonlinear dynamics of the system.
2) **Sliding Mode (Bart and Kevin)**. From simulations, we concluded that the feedback linearization version was extremely sensitive to parameter variation. Thus, we design a sliding mode controller so that the resulting feedback system is robust.

A note: **although we were able to design and implement a Jacobi Linearization controller from our physical system, we were unable to simulate the feedback linearization version. Both authors tried to design and simulate the controller independently, but the MATLAB simulation was very slow. As a result, for the nonlinear controllers, we resorted to using a model from the ME237 Spring 2008 homework sets.**

The organization of this report is: in the next section, we identify the system model. Section III shows that the open loop system is unstable via simulation. Section IV is the Jacobi linearization section. Section V has details on the nonlinear control design. We first study stability via Jacobi linearization, derive results for accessability and distinguishability for our emperical model. However, as we mentioned, MATLAB would just not terminate for the feedback linearization controller. Therefore, we just derive the equations for feedback linearization, we have only attached simulation results for the feedback linearization case since this was already part of homework assignment(s) for ME237 Spring 2008. However the point of showing some trajectory plots is to illustrate the extreme sensitivity of feedback linearization to changes in model parameters. Hence we introduce the ME237 magnetic levitation model and then design **two** sliding mode controllers for this model. One is based on the technique learned in class. The other is a technique which one of the authors found while doing background research on the topic [5]. We conclude the report with Acknowledgments and Future Work. The Appendices have the MATLAB and Mathematica code used in this report. A note on the conventions used in this report: important equations have been boxed. Also, detailed derivations have been avoided, only the concepts, final results and simulations are shown. Please contact the author of the corresponding sections for questions relating to derivations.

II. System Identification (BART AND KEVIN)

A. Physical Description of the System Components (KEVIN)

In Fig. 1, the different components are:

- $H_{Voltage\ to\ Current\ Inverter}$: This subsystem converts the output voltage from our controller into input current for the electromagnet (the transfer function was emperically determined to be $-1\frac{A}{V}$). The reason for using this system is to separate the power amplifier (for a high current sink like the electromagnet) from the controller.
- $H_{Sensor\ Electronics}$: For our Jacobi Linearization based controller, we use a simple red LED and a photoresistor. The reason for this is we emperically found that we need large gain values for the Hall Effect sensor solution. But for our nonlinear controllers, in order to sense a wider range of motion for the ball, we obtained data from two Melexis [4] Hall Effect sensors $s1$ and $s2$. We need two sensors instead of one because we emperically determined that the reading from a single sensor is saturated by the magnetic field of the electromagnet. We are able to subtract the readings of the two sensors to get a nonlinear voltage function for the position of the ball.
- $Electromagnet$: This is our plant, the model is derived below.
- $Controller$: Designing this subsystem was the goal of this project. Again, as stated earlier, we were only able to design and implement the Jacobi linearization version.

B. Mathematical Model of the Plant (BART)

The mathematical model for the electromagnet can be derived from Newton's laws and Kirchoff's laws [1]. Using Newton's laws of motion at a point y:

$$m\ddot{y} = mg - F(y, I) \tag{1}$$

Here, m is the mass of the object to be levitated (in kilograms), g is the acceleration due to gravity ($9.8m/s^2$) and F is the force exerted by the electromagnet as a nonlinear function of y (metres) and I (amperes). For simplicity, we ignored the contribution due to $\dot{y}$. In addition to the mechanics, the electromagnet also has electrical dynamics because there is a delay in the inductor reaching the steady state value of the input current. This can be easily modeled using a Norton equivalent RL circuit with u as the input current from the voltage to current inverter, R as the winding resistance, L as the electromagnet inductance and I as the current flowing through the electromagnet as a function of time:

$$uR = IR + L\frac{dI}{dt} \tag{2}$$

From basic electromagnetics, we know the force exerted due to an electromagnet at a point along its axis is of the form $f = k\frac{I^2}{y^2}$. Using this equation as a starting point, let us assume the $F(y, I)$ in (1) to be of the form:

$$F(y, I) = \frac{I^2}{b_0 + b_1 y + b_2 y^2 + by^3} \tag{3}$$

In order to determine the coefficients of the denominator in the polynomial above, we used a current value I_δ that would render the net forces to be zero at a position y_δ. Thus, (1) becomes:

$$\frac{I_\delta^2}{mg} = b_0 + b_1 y_\delta + b_2 y_\delta^2 + b_3 y_\delta^3 \tag{4}$$

Once we experimentally determined (I_δ, y_δ), we used the **polyfit** function in MATLAB to obtain:

$$F(y, I) = \frac{I^2}{-4330.7 + 2252.9y - 382.3y^2 + 21.5y^3} \tag{5}$$

Fig. 2 shows a plot of the polynomial function along with the data points. The MATLAB code for generating the plot is given in the Appendix. Note that the code also has our measured data from the physical plant. The "x"s in the figure refers to additional measurements we made two days after we obtained the polynomial function. Notice the close agreement between our system model and the new measurements. Defining $\mathbf{x} = (y, \dot{y}, I) = (x_1, x_2, x_3)$, using (1),(2) and (5), we obtain the following state space model of our system:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})u & (6)\\ y &= h(\mathbf{x}) \\ &= x_1; & (7) \end{aligned}$$

where

$$\boxed{\mathbf{f}(\mathbf{x}) = \begin{pmatrix} x_2 \\ g - \frac{x_3^2}{m(-4330.7+2252.9x_1-382.3x_1^2+21.5x_1^3)} \\ \frac{-R}{L}x_3 \end{pmatrix}} \tag{8}$$

and

$$\boxed{\mathbf{g}(\mathbf{x}) = \begin{pmatrix} 0 \\ 0 \\ \frac{R}{L} \end{pmatrix}} \tag{9}$$

We emperically determined $R = 1.7\ \Omega$, $L = 153\ mH$ and $m = 1.3 \cdot 10^{-3}\ kg$. g is defined as $9.8\ m/s^2$.

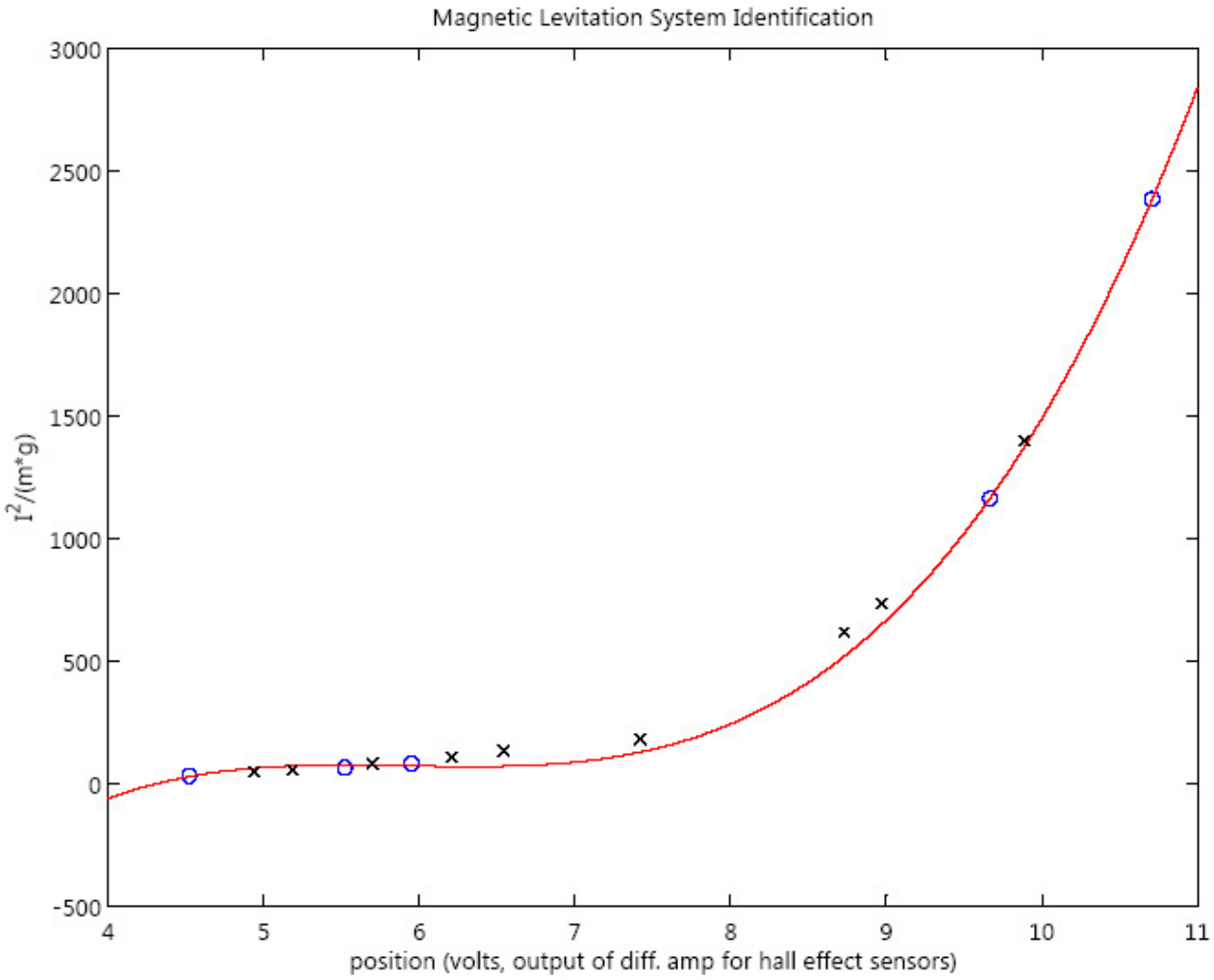

Fig. 2: Results of the polyfit function from MATLAB.

III. Open loop simulation results for the Plant (BART)

Fig. 3 and Fig. 4 shows simulation results using NelinSys [8]. Notice that the system is unstable, which implies that we need to use feedback control to stabilize the system.

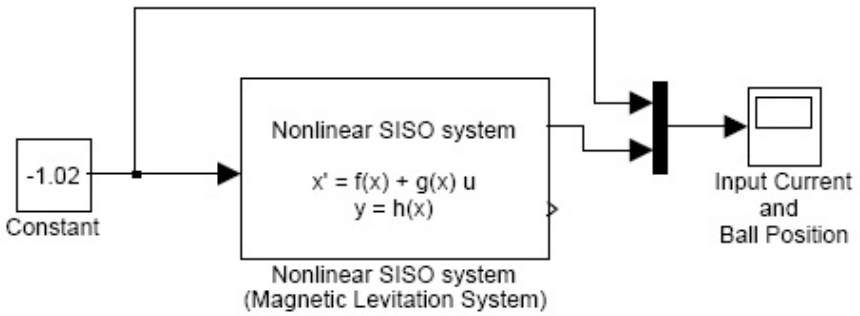

Fig. 3: Simulink model from NelinSys. We use a constant input of -1.02 amperes. Ideally, the object should levitate and the voltage output from the sensor should be constant at 5.95 volts

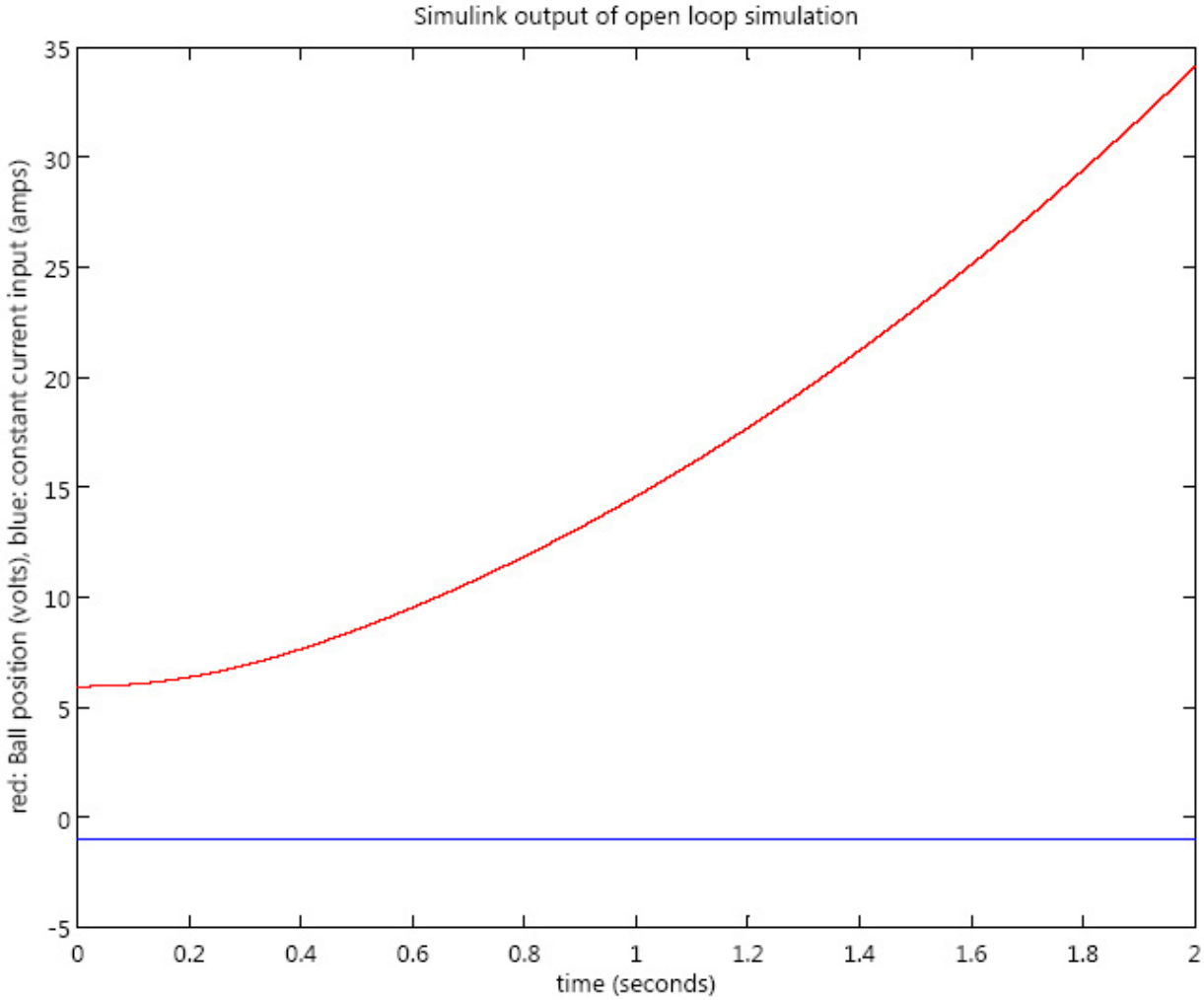

Fig. 4: MATLAB output as a result of step input to the nonlinear model

IV. Linear Controller via Jacobi Linearization (KEVIN AND BART)

A. Controller Design (KEVIN)

Recall that for avoiding huge values of sensor gain, we resorted to using a LED and photoresistor solution for the Jacobi Linearization. Following the same procedure as the previous section, the force exerted by the coil was measured over a range of different currents and positions (Fig. 5). The relationship between the ball position and voltage output of the light sensor was also measured (Fig. 5). From this data, a linearized model was determined that fit the form given above.

$$m\ddot{x} = .04 \cdot i + 26 \cdot x \tag{10}$$

$$v = 600 \cdot x \tag{11}$$

Combining the above expressions and taking the Laplace Transform, we get the following transfer function for the system:

$$H(s) = \frac{1472.4}{s^2 - 1595.04} \tag{12}$$

This system is clearly unstable, due to the existence of a right half plane pole. Furthermore, the root locus plot shows simple proportional control will not be able to stabilize the system. By looking at the Nyquist plot of the system, we can determine that a lead compensator is necessary. The Kp of the system

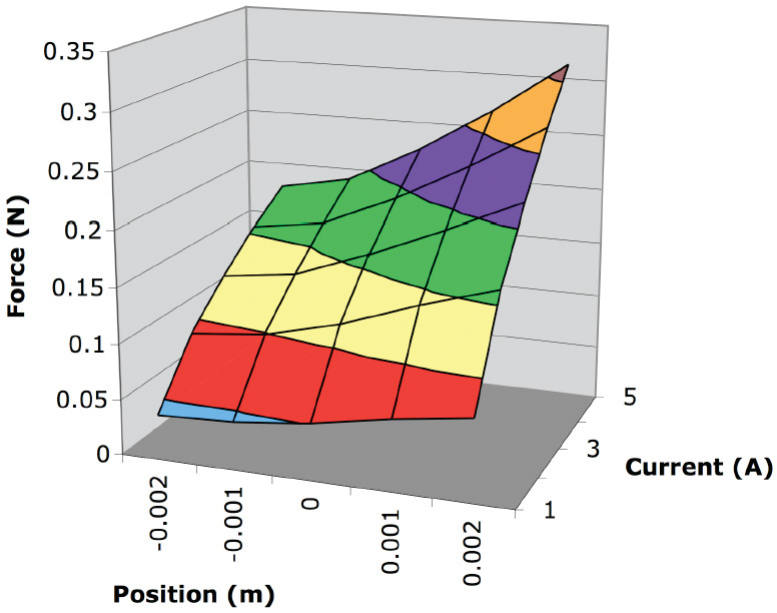

Fig. 5: Force on Ball from Electromagnet

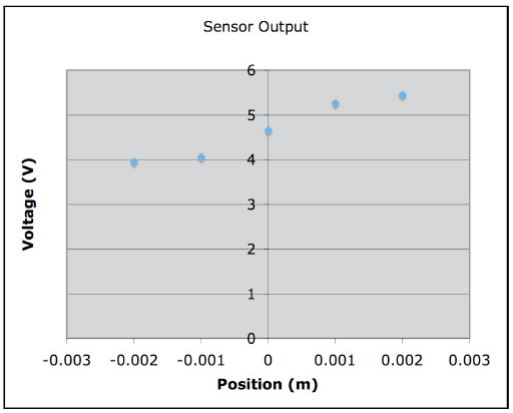

Fig. 6: Excel plot of sensor data

is chosen such that we get 1A of current for a 1mm change in position. The lead compensator is designed to give a phase margin of 60 degrees, as shown below in the Bode Plot.

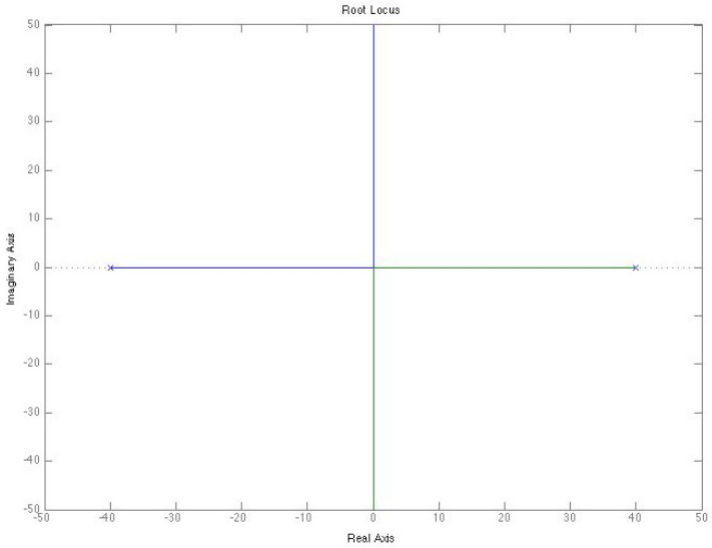

Fig. 7: Root Locus plot of our open loop system, showing RHP pole

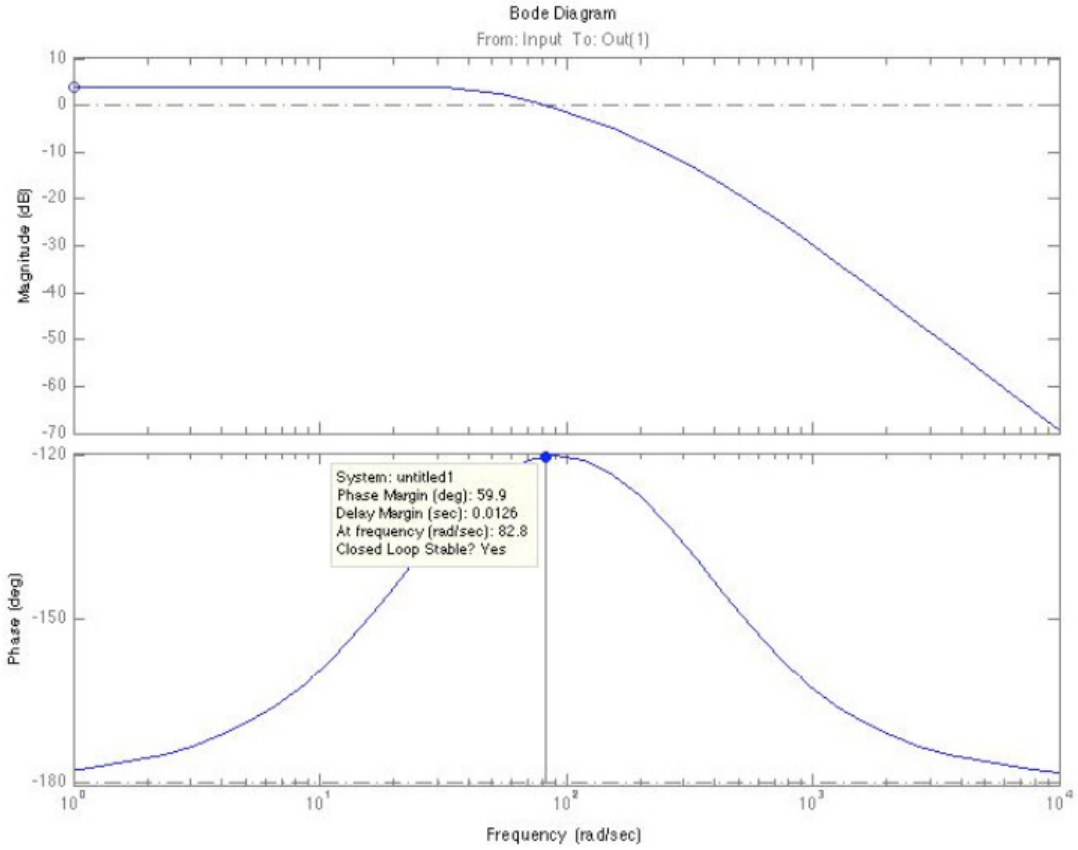

Fig. 8: Bode plot for our closed loop system

The compensator is:

$$\boxed{H_c(s) = 1.67 \cdot \frac{1 + \frac{s}{24}}{1 + \frac{s}{340}}} \tag{13}$$

B. Implementation Results (BART)

An analog circuit that implements (13) is given in Fig. 9. A picture of the ball levitating from our physical system is shown in Fig. 10. Emperically, we found that our controller was able to dampen 0.1 cm disturbances from the equilibrium point (in the vertical direction). Thus, we are able to satisfy our control goal for the Jacobi Linearization controller.

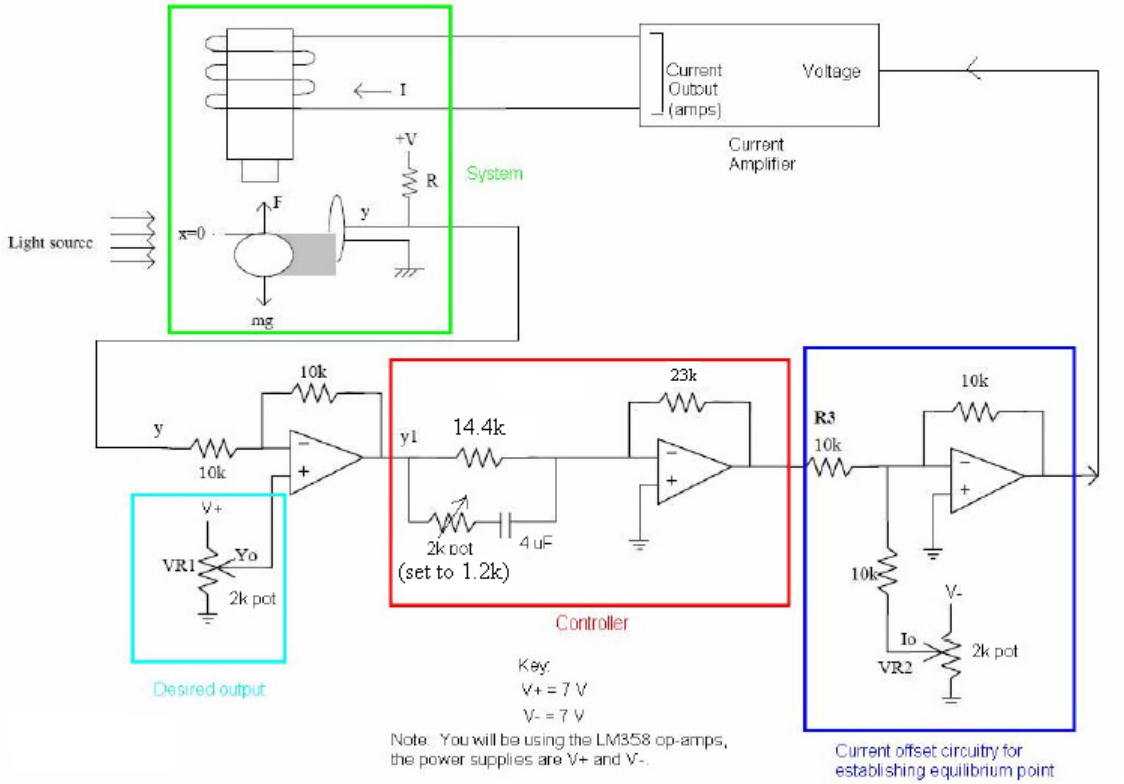

Fig. 9: Analog Circuit Implementation of our Jacobi Linearized Feedback Controller

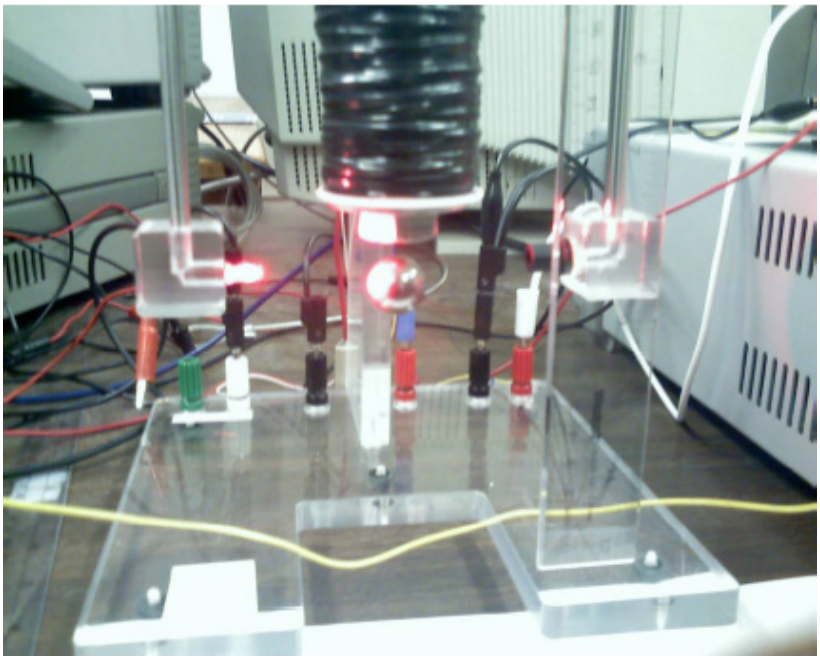

Fig. 10: Picture of the spherical ball levitating in the physical system

V. NONLINEAR CONTROLLER (BART AND KEVIN)

In this section, we design the nonlinear controllers for trajectory tracking.

A. Stability Analysis (BART)

Performing a simple Jacobi linearization of (8) we get:

$$\begin{pmatrix} 0 & 1 & 0 \\ \frac{61.3497\left(2552.9-764.6x_1+64.5x_1^2\right)x_3^2}{\left(-4330.7+2552.9x_1-382.3x_1^2+21.5x_1^3\right)^2} & 0 & -\frac{122.699x_3}{-4330.7+2552.9x_1-382.3x_1^2+21.5x_1^3} \\ 0 & 0 & -11.11 \end{pmatrix} \tag{14}$$

Evaluating (14) about the operating point $(5.95\ V, -1.02\ A)$, we have:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0.13 & 0 & 0.099 \\ 0 & 0 & -11.11 \end{pmatrix} \tag{15}$$

A quick computation of the eigenvalues for the above matrix gives $0.36, -0.36\ and\ -11.11$. Thus, we have an eigenvalue in the RHP, therefore the system is open-loop unstable. Notice that we don't have eigenvalues on the $j\omega$ axis, therefore we did not have to use fancy techniques like the Central Manifold Theorem.

B. Accessability (BART) and Distinguishability (KEVIN)

1) Accessability: We need to check if $C = [g, [f, g], [f, [f, g]]]$ is full rank. Using Mathematica, C evaluated at the equilibrium point is:

$$\begin{pmatrix} 0 & 0 & 11.11 \\ 0 & -0.88 & 123.457 \\ 0.88 & -9.96 & 1371.74 \end{pmatrix}$$

C is thus full rank, hence our system is accessible.

2) Distinguishability: We need to check if the Jacobian of O, defined below is full rank .

$$O = \begin{pmatrix} h \\ L_f h \\ L_f^2 h \end{pmatrix} \tag{16}$$

The Jacobian at the equilibrium point is:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.0079 & 0 & 0.0792 \end{pmatrix}$$

Notice that it is full rank as well, thus the system is distinguishable.

C. Control Design via Feedback Linearization (KEVIN)

Feedback linearization is a useful technique for transforming a nonlinear control problem into a simple linear problem. Unlike Jacobi linearization the nonlinearities of the system are cancelled, instead of ignored, greatly improving the performance and valid range of the controller. To implement feedback linearization, the output is differentiated until the input appears. This process is illustrated in (17)

$$\begin{aligned} y = h(\mathbf{x}) &= \dot{L_f}h \\ \dot{y} &= L_f^1 h \\ \ddot{y} &= L_f^2 h \\ \cdots & \\ y^{(r)} = L_f^r h + L_g(L_f^{r-1}h)u &= v \\ u &= \frac{1}{L_g(L_f^{(r-1)})}(-L_f^r h + v) \\ z &= \begin{pmatrix} y \\ \dot{y} \\ . \\ . \\ . \\ y^{(r-1)} \end{pmatrix} \end{aligned} \tag{17}$$

1) Feedback linearization equations for our model (KEVIN): The relative degree (r) of the magnetic levitation system is 3, which gives us no internal dynamics to worry about. For our model stated earlier in this report, the new states are given in (18)

$$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ g - \frac{x_3^2}{m\cdot(a_0+a_1x_1+a_2x_1^2+a_3x_1^3)} \end{pmatrix} \tag{18}$$

By setting y^r equal to a synthetic input v, and solving for the system input u, we obtain an expression for the input that linearizes the system.

$$u = \frac{1}{\frac{R}{L}\frac{2x_3}{m(a_0+a_1x_1+a_2x_1^2+a_3x_1^3)}}\left(\frac{-x_3^2}{m}\left(\frac{x_2(a_1+2a_2x_1+3a_3x_1^2+\frac{2R}{L}(a_0+a_1x_1+a_2x_1^2+a_3x_1^3))}{a_0+a_1x_1+a_2x_1^2+a_3x_1^{3^2}}\right)\right)+v$$

Our transformed system is of the form $\dot{z} = Az + Bv$, with A and B given below. The input v for the new system can be obtained by any linear design method such as pole placement, LQR, etc.

$$\begin{aligned} A &= \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \\ B &= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \end{aligned} \tag{19}$$

2) Feedback simulation results for ME237 HW problem (KEVIN): We attempted to simulate this system using Matlab, but due to the complexity, the numerical methods Matlab uses were not converging. Because of this problem, the magnetic suspension model given in the homework for ME237 [7] was used whenever Matlab simulations were performed. In order to avoid repeating the homework solutions, we have just given the results here.

Through simulation using Matlab, we can see that feedback linearization can be used to stabilize the system, and allow tracking of a reference input. To test robustness, the resistance is varied around the nominal position. By varying the resistance by $.1\%$, the system no longer tracks the desired trajectory exactly. If the resistance is varied by greater than this amount, the system can become unstable. To address the problem of robustness, a different type of controller is used. This is the topic of the next section.

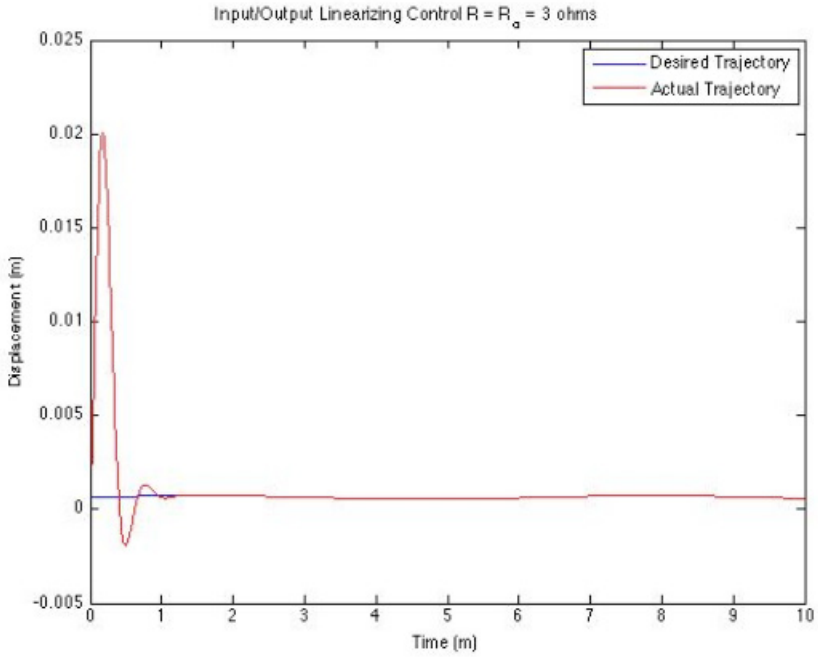

Fig. 11: Feedback linearization simulation results of ME237 HW problem with nominal R

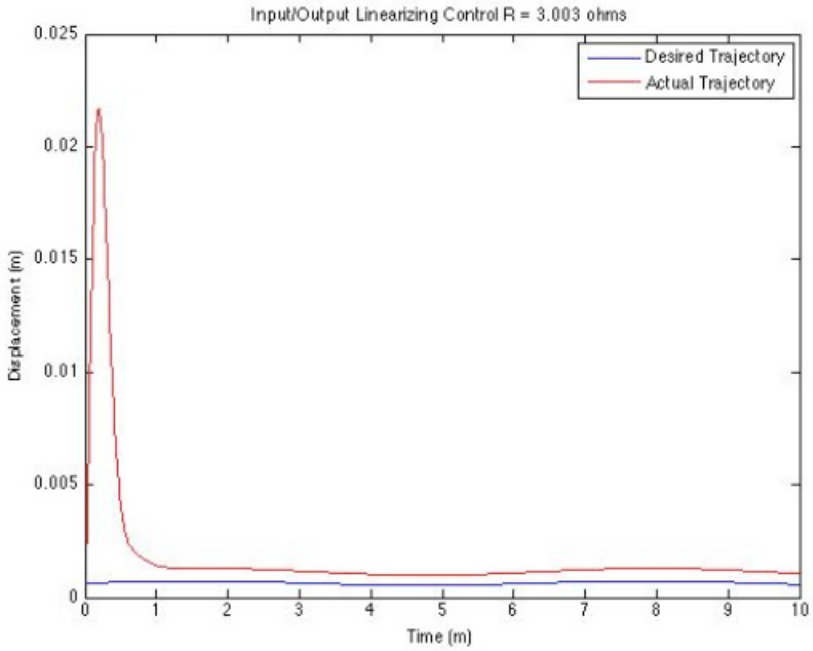

Fig. 12: Feedback linearization simulation results of ME237 HW problem with a small change in R

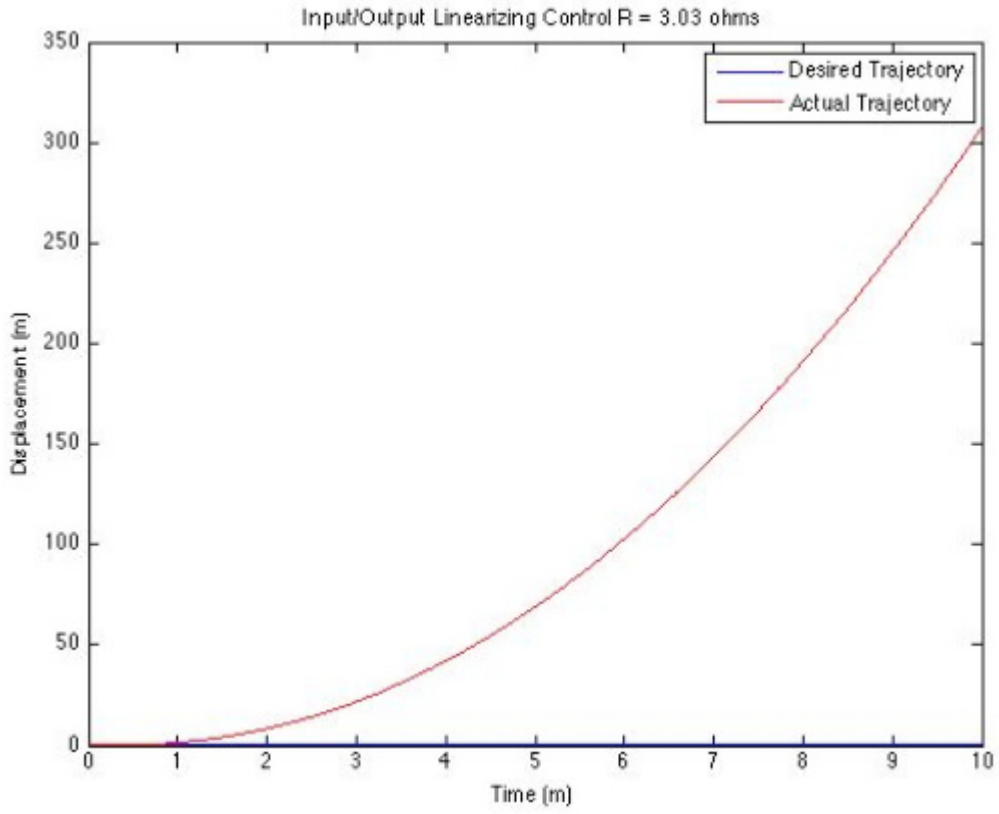

Fig. 13: Notice that $R = 3.03\ \Omega$ causes instability

D. Control Design via Sliding Mode Control (BART AND KEVIN)

1) Model of Magnetic Levitation System from ME237 [7]: Since we were unable to simulate the feedback linearization controller for the emperical model, we switched to the model from ME237,

summarized in (20)

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= g - \frac{\alpha z^2}{m(2x+\beta)} \\ \dot{z} &= \frac{2x+\beta}{\alpha}(u - Rz) + \frac{2yz}{2x+\beta} \end{aligned} \tag{20}$$

Here $\alpha = 2.56 \cdot 10^{-4}$, $\beta = 5.28 \cdot 10^{-4}$ and $g = 9.8\ m/s^2$. A picture of this system is shown in Fig. 14. The equilibrium points under consideration are $x = 6.35 \cdot 10^{-4}\ m, y = 0\ m/s^2, z = 1.05\ A$. Again, these computations were part of the ME237 homework and are not repeated here.

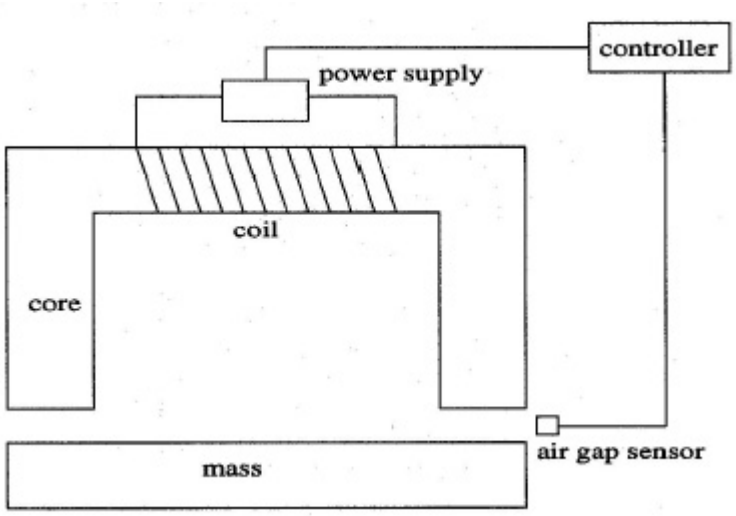

Fig. 14: The Magnetic Levitation model from ME237

To address the issue of robustness from the feedback linearization section, two sliding mode controllers have been designed for the system above.

2) Sliding Mode Control based on technique learned in class (KEVIN): Since the relative degree of the system was 3, the sliding surface was chosen to be:

$$s = (\frac{d}{dt} + \lambda)^2 \tilde{x} \tag{21}$$

with $\tilde{x} = x_1 - x_{1d}$ and λ chosen to be 30 to provide the desired performance characteristics. If the surface s goes to zero, we will have the position go to the desired value asymptotically. To ensure that s will go to zero, we need the condition:

$$s\dot{s} \leq -k\frac{s^2}{\phi} \tag{22}$$

To satisfy this condition, we find $\dot{s}$ and set it equal to $\frac{-k \cdot s}{\phi}$. We then solve for the output u. The final expressions are shown below.

$$\boxed{s = g - \frac{\alpha x_3^2}{m(2x_1+\beta)^2} - \ddot{x_{1d}} + 2\lambda(x_2 - \dot{x_{1d}}) + \lambda^2(x_1 - x_{1d})} \tag{23}$$

$$\boxed{u = \frac{1}{\frac{2x_3}{m(2x_1+\beta)}} \left[\frac{2Rx_3^2}{m(2x_1+\beta)} - \ddot{x_{1d}} + 2\lambda(g - \frac{\alpha x_3^2}{m(2x_1+\beta)} - \ddot{x_{1d}}) + \lambda^2(x_2 - \dot{x_{1d}}) + ks \right]} \tag{24}$$

ϕ is the value of the boundry layer. We are guaranteed that the system will reach the boundry layer in a finite time, and will remain within the layer for all time, but have no guarantees on the value of the system within the layer. Therefore, we choose ϕ to be small so we get good convergence to the desired value. The value of k is chosen to be $\eta + F$, where η is some safety factor on the system, and F is the bound on the model uncertainty. For our model, we assume that the coil resistance is known to within 10% of the actual value.

$$F = |\frac{2x_3^2}{m(2x_1 + \beta)} \cdot 0.1| \tag{25}$$

We set $\eta = 10$ to ensure quick convergence to the boundary layer. This controller was simulated using Matlab, and showed very good performance. The system exhibits excellent tracking when the resistance is the nominal value. It also shows good robustness characteristics, as good tracking is still achieved when the resistance is varied up to 10% from the nominal value. This shows a marked improvement over the feedback linearization controller, which could not handle large changes in resistance.

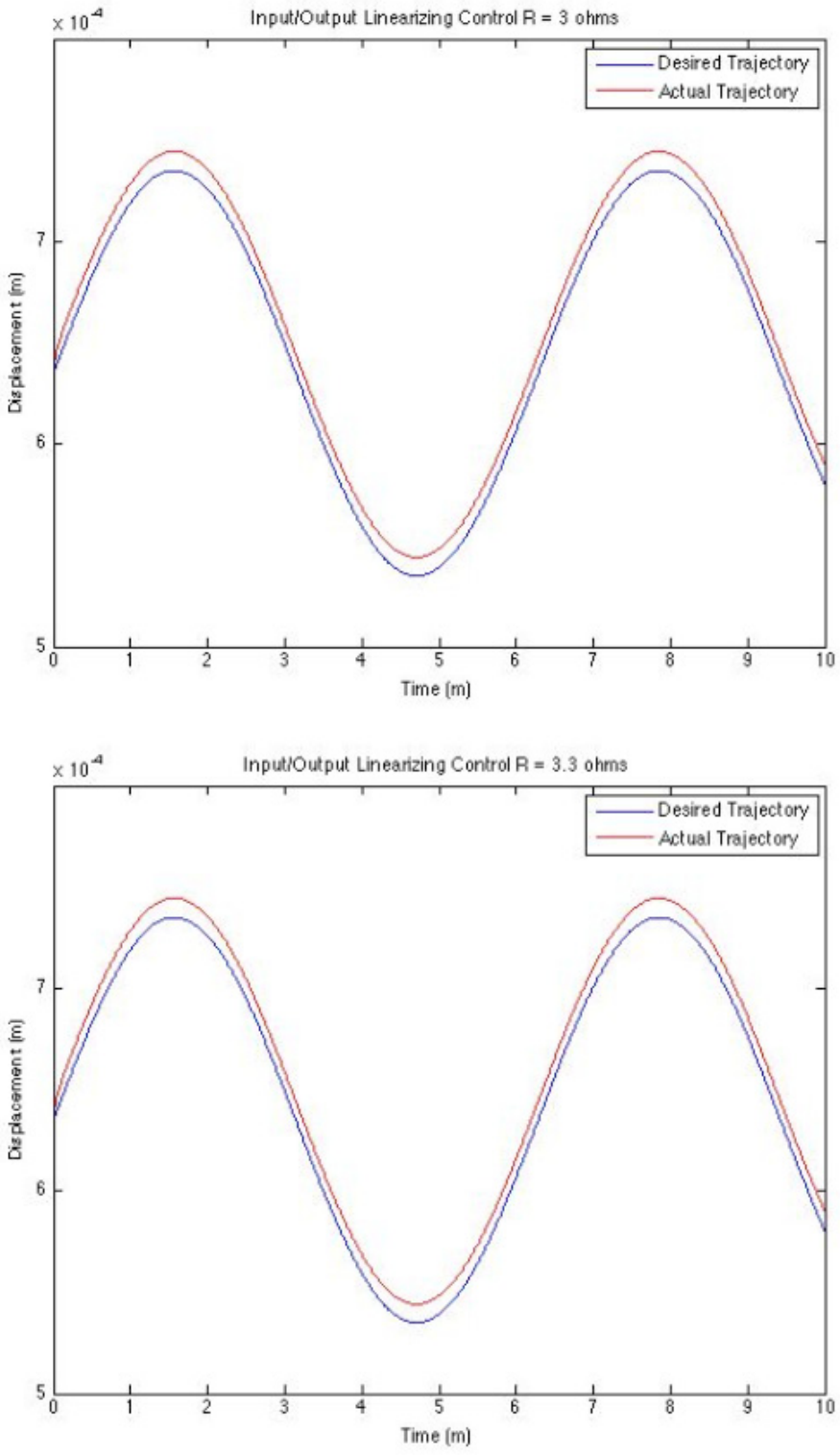

Fig. 15: Simulation results of the sliding mode controller with boundary layer. Notice how this controller is insensitive to parameter changes, unlike the feedback linearization case.

3) Sliding Mode Control based on technique in [5] (BART): In this section, we used the following simple transformation from [5] for the system in (20)

$$\begin{aligned} z_1 &= x - x_d \\ z_2 &= y \\ z_3 &= g - \frac{\alpha z^2}{m(2x+\beta)} \end{aligned} \tag{26}$$

Using (26) in (20), we get the following set of equations:

$$\begin{aligned} \dot{z_1} &= z_2 \\ \dot{z_2} &= z_3 \\ \dot{z_3} &= f(\mathbf{z}) + g(\mathbf{z}) \cdot u \end{aligned} \tag{27}$$

Notice that (27) is in the nonlinear canonical form, where:

$$\begin{aligned} \mathbf{f}(\mathbf{z}) &= \begin{pmatrix} z_2 \\ z_3 \\ \frac{2R(g-z_3)}{\alpha} \end{pmatrix} \\ \mathbf{g}(\mathbf{z}) &= \begin{pmatrix} 0 \\ 0 \\ \frac{1}{m(2(z_1+x_d)+\beta)} \end{pmatrix} \end{aligned} \tag{28}$$

Now, proceeding as in [5], let us define our sliding surface to be:

$$\boxed{s = g - \frac{\alpha z^2}{m(2x+\beta)^2} + \lambda_1 y + \lambda_2 (x - x_d)} \tag{29}$$

Notice the subtle differences between (29) and (23). Because of our transformation, we have been able to eliminate $\ddot{x}$ from our sliding surface. Defining our sliding condition [9] to be:

$$s\dot{s} \leq -k|s| \tag{30}$$

we get the following expression for the input:

$$\boxed{u = m \cdot (2x+\beta)\left[-\frac{2Rz^2}{m(2x+\beta)} + \lambda_1 \dot{y} + \lambda_2(\dot{x} - \dot{x_d})\right]} \tag{31}$$

This controller is also insensitive to parameter changes. Simulation results for R_0 and a 5% variation in R have been simulated.

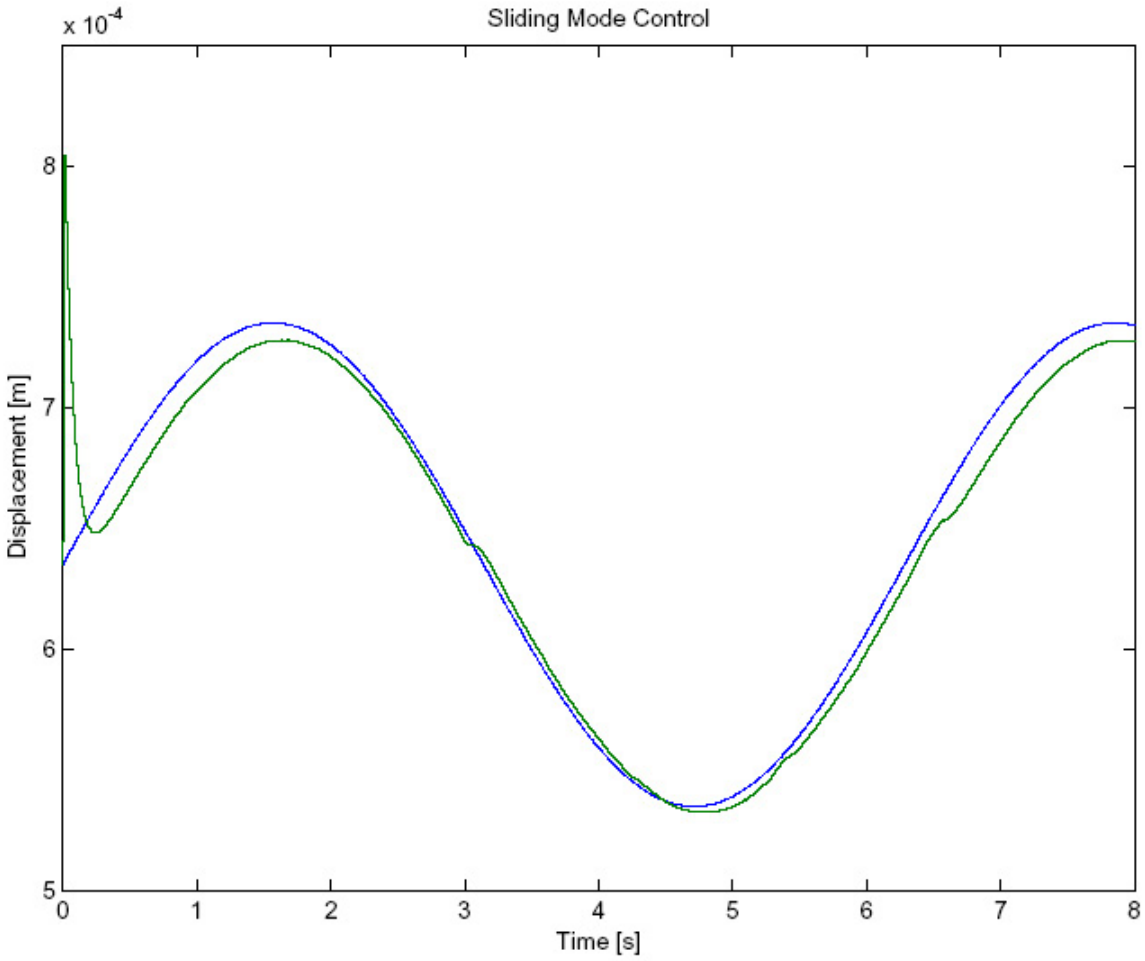

Fig. 16: Simulation results of the sliding mode controller without boundary layer (R_0)

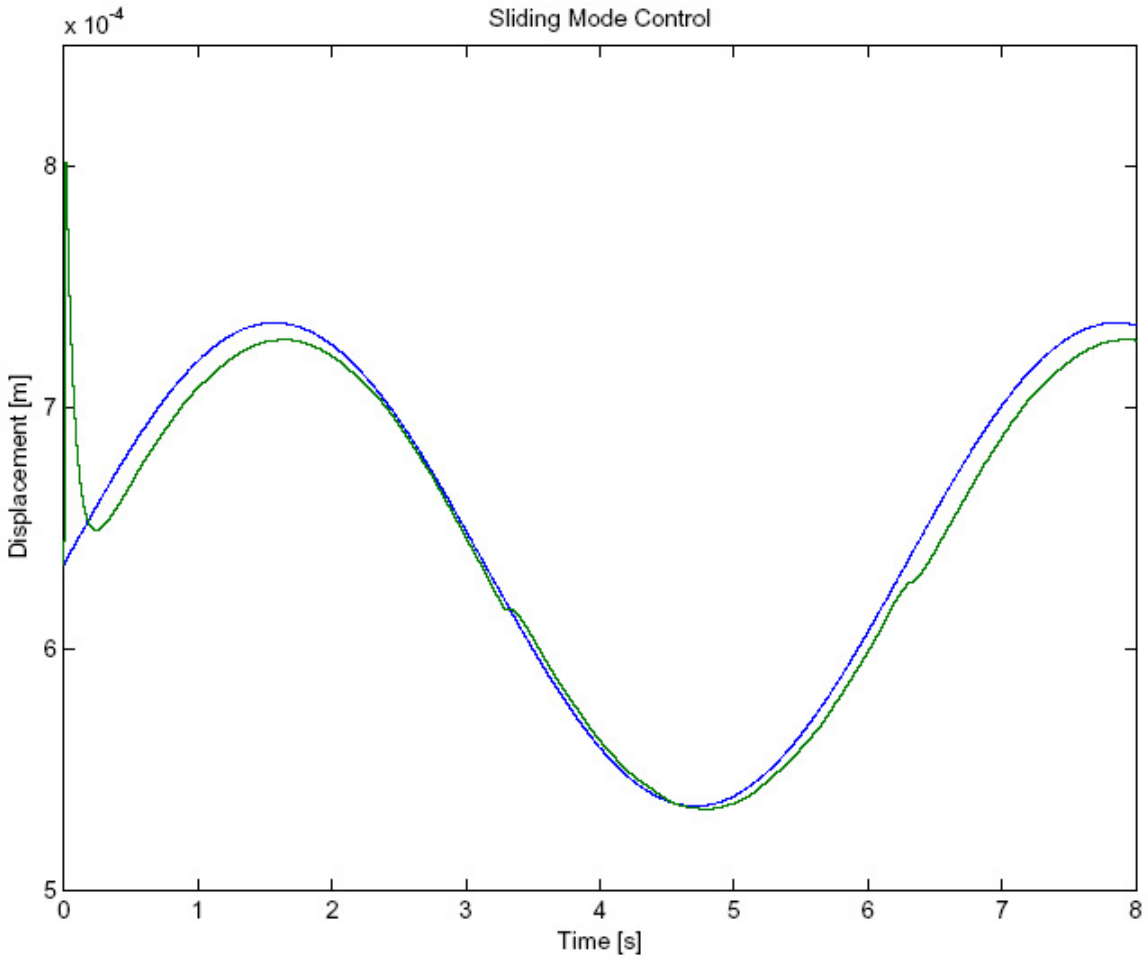

Fig. 17: Simulation results of the sliding mode controller without boundary layer (R)

VI. Conclusions and Future Work

In this report, a variety of controllers based on Jacobi Linearization, Feedback Linearization and Sliding mode control have been presented. Future work includes:

1) Understand why our emperical magnetic levitation model is not simulating under MATLAB, correct and refine model.
2) Implement the controllers on a digital platform, like Real Time Windows Target [3]. The authors tried to implement the Jacobi linearization version on the digital platform but without any success.

Acknowledgment

Many thanks to Prof. Hedrick and Giovanni for helping us throughout ME237 Spring 2008. Comments and support from Ferenc, Tho and Winthrop from the Electronic Support Group in the EECS department at UC Berkeley have been invaluable.

References

[1] A. E. Hajjaji and M. Ouladsine, "Modeling and nonlinear control of magnetic levitation systems," *IEEE Transactions on Industrial Electronics*, vol. 48, no. 4, pp. 831–838, August 2001.
[2] H. K. Khalil, *Nonlinear Systems*. New Jersey: Prentice Hall, 1996.
[3] Mathworks. (2008, February) Matlab real time windows target module. [Online]. Available: http://www.mathworks.com/products/rtwt/
[4] Melexis. (2008, March) Programmable linear hall effect sensor. [Online]. Available: http://www.melexis.com/prodfiles/0004758_MLX90215_Rev007.pdf

[5] N.F.Al-Muthairi and M.Zribi, "Sliding mode control of a magnetic levitation system," *Mathematical Problems in Engineering*, vol. 2, pp. 93–107, 2004.
[6] U. of California Berkeley. (2006, September) EE128: Feedback systems. [Online]. Available: http://inst.eecs.berkeley.edu/~ee128
[7] ——. (2008, May) Me237 spring 2008 homepage. [Online]. Available: http://www.me.berkeley.edu/ME237
[8] M. Ondera. (2008, March) Nonlinear control systems toolbox for matlab. [Online]. Available: http://www.melexis.com/prodfiles/0004758_MLX90215_Rev007.pdf
[9] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*. New Jersey: Prentice Hall, 1991.

APPENDIX
MATLAB CODE FOR SYSTEM IDENTIFICATION (BART AND KEVIN)

```
% Bharathwaj Muthuswamy and Kevin Peterson
% mbharat@cory.eecs, kevincp@eecs
% EE128 Magnetic Levitation Setup nonlinear model
% Spring 2008
clear
% Measured equilibrium values of current (output of
% voltage to current inverter) and position (output of
% difference amplifier for the two hall effect sensors)
% The underscore g notation means the values below are equilibrium
% values
Vy_g = [4.52 5.52 5.95 9.67 10.71];
I_g  = [-0.63 -0.91 -1.02 -3.85 -5.51];
% hex nut weights 1.3 grams, remember to convert to kg
m = 1.3e-3;
% acceleration due to gravity (duh)
g = 9.8;
% system identification:  try a cubic first.  Maybe need to do
% a couple of iterations before we get the best fit poly
Y = (I_g).^2/(m*g);
P = polyfit(Vy_g,Y,3)
% plot the data points
plot(Vy_g,Y,'o');
title('Magnetic Levitation System Identification');
xlabel('position (volts, output of diff. amp for hall effect sensors)');
ylabel('I^2/(m*g)')
% plot the cubic
hold;
N = [4:0.1:11];
px = P(1).*N.^3 + P(2).*N.^2 + P(3).*N + P(4);
plot(N,px,'r')
% now plot a second set of measured data points (Vy_g,Iy_g) and see how
% close they lie to the best fit poly
Vy_g_test = [4.94 5.18 5.70 6.21 6.54 7.42 8.73 8.97 9.89];
I_g_test = [-0.78 -0.84 -1.02 -1.18 -1.31 -1.52 -2.8 -3.06 -4.22];
plot(Vy_g_test,(I_g_test.^2)/(m*g),'kx');
% points look really good.
% FYI the best fit poly from the test data points was:
% Ptest = 17.9*y^3 - 318.5*y^2 + 1931.3*y - 3889.3
% The poly from our data points (to be used in the model is):
% P = 21.5*y^3 - 382.3*y^2 + 2252.9*y - 4330.7
```

APPENDIX
MATLAB CODE FOR FEEDBACK LINEARIZATION CONTROL (KEVIN)

```
function xprime = maglev(t, x)

%Constants
g = 9.81;
```

```
R0 = 3;       %nominal--used in controller
R = 3.03;   %actual physical value
M = 8.91;
mu0 = 1.26e-6;
mu1 = 1e4;
mu2 = 100;
L1 = .28;
L2 = .05;
A = 25e-5;
N = 800;
alfa = mu0*N^2*A;
beta = L1/mu1+L2/mu2;

%Calculate desired trajectory for tracking controller.
hd = 6.35e-4+1e-4*sin(t);
hd1 = 1e-4*cos(t);
hd2 = -1e-4*sin(t);
hd3 = -1e-4*cos(t);


%Transform to linearized states
xb = 2*x(1)+beta;

%Transform States
z = [x(1); x(2); g - alfa*x(3)^2/(M*xb^2)];

Lf3h = 4*alfa*x(3)^2/M/xb^3*x(2) \\
 - 2*alfa*x(3)/M/xb^2*((-xb)*R0*x(3)/alfa+2*x(2)*x(3)/xb);

LgLf2h = -2*x(3)/M/xb;

%Calculate control action
v = 450*[3.6 .6 .22^2]*[hd - z(1); hd1 - z(2); hd2 - z(3)];
u = 1 / LgLf2h * (-Lf3h + v);


%Equations of motion for plant given in HW2
xprime(1,1) = x(2);
xprime(2,1) = g-alfa*x(3)^2/M/xb^2+Fd;
xprime(3,1) = xb/alfa*(u-R*x(3))+2*x(2)*x(3)/xb;


%State derivative of integral of u.  This is a hack to get
%ode45 to store the input u for us.  If we want to plot u
%later, we do the following
%    dx=diff(x(:,4))
%    dt=diff(t)
% Then u = dx./dt and we can:   plot(t(2:end),u);
xprime(4,1) = u;
```

APPENDIX
MATLAB CODE FOR SLIDING MODE CONTROLLER (KEVIN)

```
function xprime = maglev(t, x)

%Constants
g = 9.81;
R0 = 3;       %nominal--used in controller
R = 3.3;   %actual physical value
Rchange = R0 * .1;     %assume R varies 10%
M = 8.91;
mu0 = 1.26e-6;
mu1 = 1e4;
mu2 = 100;
L1 = .28;
L2 = .05;
A = 25e-5;
N = 800;
alfa = mu0*N^2*A;
beta = L1/mu1+L2/mu2;

lamda = 30;
eta = 10;
phi = .01;

t

%Calculate desired trajectory for tracking controller.
hd = 6.35e-4+1e-4*sin(t);
hd1 = 1e-4*cos(t);
hd2 = -1e-4*sin(t);
hd3 = -1e-4*cos(t);


%Transform to linearized states
xb = 2*x(1)+beta;

F = abs(2*x(3)^2*Rchange / (M*xb));
k = eta + F;

s = g - alfa*x(3)^2 / (M*xb^2) - hd2 + 2*lamda*(x(2) - hd1) \\
 + lamda^2*(x(1) - hd);

u = (M*xb / (2*x(3))) * (2*R0*x(3)^2/(M*xb) - hd3 + \\
2*lamda*(g - alfa*x(3)^2/(M*xb^2) - hd2) + lamda^2*(x(2) - hd1) + k*s/phi);


%Equations of motion for plant given in HW2
xprime(1,1) = x(2);
xprime(2,1) = g-alfa*x(3)^2/M/xb^2;
```

```
xprime(3,1) = xb/alfa*(u-R*x(3))+2*x(2)*x(3)/xb;


%State derivative of integral of u.  This is a hack to get
%ode45 to store the input u for us.  If we want to plot u
%later, we do the following
%    dx=diff(x(:,4))
%    dt=diff(t)
% Then u = dx./dt and we can:   plot(t(2:end),u);
xprime(4,1) = u;
```

APPENDIX
MATLAB CODE FOR SLIDING MODE CONTROLLER (BART)

```
function xprime = magsussm(t, x)
%Model for magnetic suspension
%Sliding Mode Controller
%ME237 Sp08 Final Project
%Bharathwaj Muthuswamy and Kevin Peterson
%This function takes the current simulation time and the
%current state as inputs, and it outputs the state
%derivates which ode45 (or other matlab ode solver) then
%integrates.


%Define constants
g = 9.81;
R0 = 3;       %nominal--used
R = 3;   %actual physical value
M = 8.91;
mu0 = 1.26e-6;
mu1 = 1e4;
mu2 = 100;
L1 = .28;
L2 = .05;
A = 25e-5;
N = 800;
alpha = mu0*N^2*A;
beta = L1/mu1+L2/mu2;

%Calculate desired trajectory for tracking controller.
%This is a good desired trajectory to start with
hd = 6.35e-4+1e-4*sin(t); %xd
hd1 = 1e-4*cos(t); %xd'

%Transform to linearized states
xb = 2*x(1)+beta;   %Just a notational convenience since
                    %we're constantly dividing by (2*x1+beta)

%Calculate control action
```

```
lambda1 = 610; %61
lambda2 = 9000; %930
k=570; %1050
% sliding surface
f1 = (2*R*x(3)^2)/(M*xb);
g1 = 1/(M*xb);
s =  g-alpha*x(3)^2/M/xb^2+lambda1*x(2)+lambda2*(x(1)-hd);
u = (1/g1)*(-f1-lambda1*(g-alpha*x(3)^2/M/xb^2)-lambda2*x(2)-k*sign(s));

%Equations of motion for plant given in HW2
xprime(1,1) = x(2);
xprime(2,1) = g-alpha*x(3)^2/M/xb^2;
xprime(3,1) = xb/alpha*(u-R*x(3))+2*x(2)*x(3)/xb;


%State derivative of integral of u.  This is a hack to get
%ode45 to store the input u for us.  If we want to plot u
%later, we do the following
%    dx=diff(x(:,4))
%    dt=diff(t)
% Then u = dx./dt and we can:   plot(t(2:end),u);
xprime(4,1) = u;
```

APPENDIX
MATHEMATICA CODE FOR ACCESSABILITY AND DISTINGUISHABILITY (BART)