ENGR3370: Magnetic Levitation

Katya Donovan and Ava Lakmazaheri

May 8, 2019

1 Introduction

Magnetic levitation (MagLev) describes the suspension of an object in air due only to the force of a magnetic field. Given the inverse relationship of force to the distance between two magnets, there is no point where the system will naturally levitate. To achieve this state, we must control the force of an electromagnet, switching polarities or alternating between on/off states.

2 System Modeling

2.1 System Overview

The system uses a solenoid with a controllable output strength to levitate a set of permanent magnets. By inputting a pulse width modulation (PWM), we can control the average voltage of the solenoid. Thus, we knew that a voltage should come out of our controller.

This voltage goes into the plant which in turn models the effect of the solenoid on our system. Since we are levitating a magnet, we are interested in its position and thus chose position to be the output of the plant.

We can check this position of the magnet using a Hall Effect sensor. Knowing that the Hall Effect sensor outputs a reading proportional to its voltage, which is in turn proportional to the strength of the magnetic field, we kept the input to the controller in units of sensor value. We converted our desired position into these sensor value units, which we fed into our closed loop system.

2.2 Plant

Modeling the solenoid as an inductor and a resistor in series, we can calculate its voltage using the following equation:

$$V(t) = R_{sol} * I(t) + L_{sol} * \frac{dI(t)}{dt} \tag{1}$$

Converting this to the Laplace domain gives:

$$V(s) = R_{sol} * I(s) + L_{sol} * s * I(s) \tag{2}$$

This results in the following transfer function:

$$\frac{I(s)}{V(s)} = \frac{1}{R_{sol} + L_{sol} * s} \tag{3}$$

Next, to calculate the force between the solenoid and the permanent magnets, we looked at the energy of the surrounding magnetic field.

$$Energy = \frac{1}{2} * L(x) * I^2 \tag{4}$$

To determine the force generated by the magnetic field, we found the derivative of its energy with respect to x.

$$F_B = \frac{E}{dx} = \frac{1}{2} * \frac{L(x)}{dx} * I^2 \tag{5}$$

The total inductance of the system is attributed to 1) the constant inductance of the solenoid and 2) an inductance that is generated by the interaction of the solenoid and the magnet, which changes inversely proportional to the distance between them.

$$L(x) = L_{sol} + \frac{L_o * X_o}{x} \tag{6}$$

Thus, equation 5 simplifies to:

$$F_B = \frac{I^2}{2} * \frac{L_o * X_o}{x^2} = C * \frac{I^2}{X^2} \tag{7}$$

where

$$C = \frac{L_o * X_o}{2} \tag{8}$$

Since the force is not a linear equation, we linearized it around the initial position and the initial current.

$$F_B = C * \frac{I_o^2}{X_o^2} + \frac{2 * C * I_o}{X_o} * I - \frac{2 * C * I_o}{X_o^3} * X \tag{9}$$

Given that we care about position, we decided its initial value and found the corresponding initial current that would put the system in equilibrium. Thus, we are only modeling the system around these initial values. We drew a free-body diagram and equated the force of gravity to the force of the magnetic field.

$$F_B = C * \frac{I_o^2}{X_o^2} = F_G = m * g \tag{10}$$

Thus for an initial position,

$$I_o = X_o * \sqrt{\frac{m*g}{C}} \tag{11}$$

In order to find the general effect of force on magnet position, we refer back to the free body diagram.

$$m * \frac{dx^2}{d^2t} = mg - C * \frac{I_o^2}{X_o^2} + \frac{2*C*I_o}{X_o} * I - \frac{2*C*I_o}{X_o^3} * X \tag{12}$$

Using equation 10, this simplifies to

$$m * \frac{dx^2}{d^2t} = -\frac{2*C*I_o}{X_o} * I + \frac{2*C*I_o}{X_o^3} * X \tag{13}$$

When we convert this to the Laplace domain, we get the following:

$$m * s^2 * X(s) - \frac{2*C*I_o}{X_o^3} = -\frac{2*C*I_o}{X_o} * I(s) \tag{14}$$

Thus, we can find the transfer function from current to position:

$$\frac{X(s)}{I(s)} = \frac{2*C*I_o*X_0^2}{2*C*I_o - X_o^3*m*s^2} \tag{15}$$

Finally, in order to find the transfer function of voltage to position, we multiply the two intermediate transfer functions to get the following:

$$\frac{X(s)}{V(s)} = \frac{2*C*I_o*X_0^2}{2*C*I_o - X_o^3*m*s^2} * \frac{1}{R_{sol} + L_{sol}*s} \tag{16}$$

This transfer function is the plant of our closed loop system.

2.3 Measured Initial Conditions

We measured the inductance and resistance of our solenoid with an LCR meter. After determining the initial position of the permanent magnet, we used the same tool to measure the initial inductance of the solenoid/permanent magnet interaction.

Parameters	Value
Initial Position	0.01 m
Initial Inductance	0.03362 H
Mass of Magnet	0.0009 kg
Inductance of Solenoid	0.03344 H
Resistance of Solenoid	25.4 Ohms

2.4 Pole-Zero Plot of Plant

The described plant has three poles on the real axis. One pole is in the right half-plane, making the system unstable. With our knowledge of the root locus, we realized that a simple proportional controller would not make this system stable, because the two right-most poles would become imaginary and unstable. Additionally, an integral controller would add a pole to the origin, making the system at best marginally stable. However, a proportional-derivative (PD) controller adds a zero to the system which, at high gains, could move our unstable pole to the left half-plane.

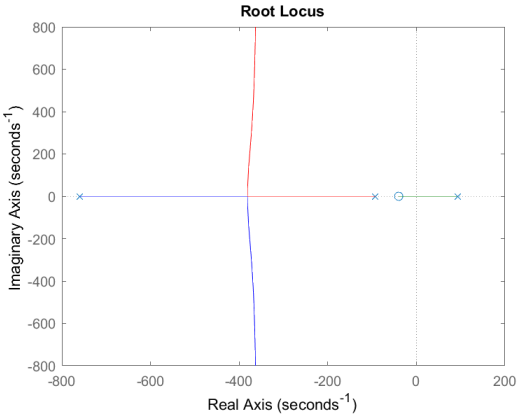

Figure 1: Open Loop Pole Zero Plot

In order to stabilize our system, we had to be intentional about the placement of this zero. It had to be in the left half-plane, but to the right of the middle pole in order to move our right-most pole. We chose the zero of the closed loop system to be -30 for these reasons.

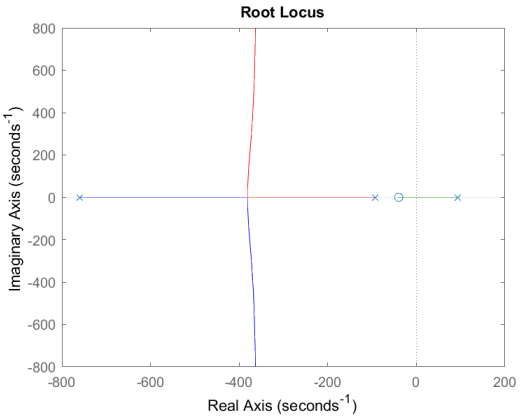

Figure 2: Root Locus of Closed Loop System

According to our model, we should place a zero at -30 and implement a gain of 298000 (see Figure 3).

2.5 Validation of Model

Before implementing this model, we wanted to simulate it to see if it behaved as we expected. We tested the simple case of using no controller, expecting the simulation to go unstable. With Simulink, we were able to quickly

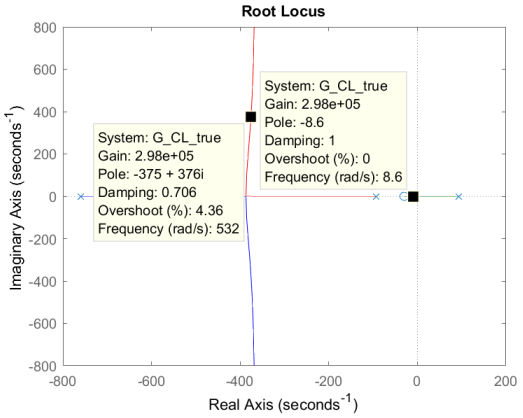

Figure 3: Expanded View of Root Locus

verify this result. Figure 4 shows that the magnet moves in the direction of the solenoid with a driving voltage and no controller. Since the model has no boundary to capture how the solenoid acts as a physical stop to the magnet, this instability goes off to infinity.

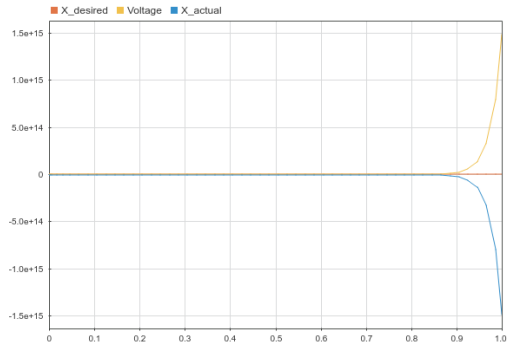

Figure 4: System Response to No Controller

3 Physical Setup

The solenoid is held by a laser-cut hardboard frame and secured with 3D-printed PLA housing. A hall effect sensor is situated directly under the electromagnet, measuring strength of its magnetic field and outputting a proportional voltage. An Arduino Uno is used to measure these values and control the voltage provided to the solenoid.

Moving forward with this design, we recognized the following physical constraints. First, the frame lacks sturdiness and is susceptible to vibration if the magnets oscillate. Second, there is a limited range in which the Hall Effect sensor can give accurate readings. If permanent magnets are placed in close proximity to the sensor (approximately 5 mm), the sensor readings saturate (see Figure 5). Third, our solenoid's maximum output is 12 Volts, limiting the maximum strength of the magnetic field. This means that if the permanent magnets drop below a certain range, they cannot be recovered. Finally, we recognize the importance of speed in sending commands to the solenoid. If the system is slow to update, the magnets may have time to fall outside of the controllable range. Given that the Arduino Uno has limited processing speed, we must be especially careful of adding print statements or computationally intensive commands.

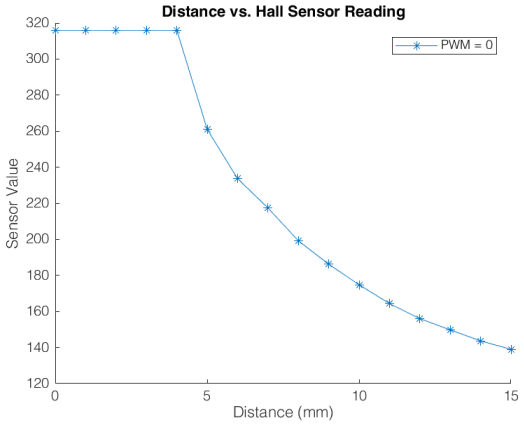

Figure 5: Effect of Distance on Hall Sensor Reading

4 Controller Implementation

Having experimentally determined a viable levitation height given the strength of the solenoid, we were easily able to map our target position to a Hall Effect sensor value. This reading from the sensor became the input of our controller.

In early tests, we realized that the sensor can undergo huge changes in magnitude in a very short time, resulting in choppy oscillations that destabilize the system. To counteract this, we used a moving average to smooth out the changes in the sensor reading. It is this smoothed value that is used to calculate the error from the target position and the resulting error rate of change.

The gains that scale these quantities were based on the previously described root locus. We placed our zero at s = -30 on the real axis by setting kp = 30 and kd = 1. In order to stabilize the system, the root locus required the k gain to be as high as 298000. However, when k reached such a large number, the system still destabilized. We theorized that this might be because the processing speed of the Arduino is not sufficiently fast to respond to such large changes in output in a short time. Alternatively, the high k gain may cause a lag that is detrimental to our time-sensitive system. Thus, we experimentally reduced k until we saw improved results.

We subtracted the output of our PD controller from a baseline duty cycle of 50%. The reason for this step is clearest when considering the system resting at its target value. Here, there will be zero error and zero error rate of change. However, if the output of the PD controller gives a 0% duty cycle, the object will fall due to its mass and the force of gravity. Thus we can see why the electromagnet must provide a constant attracting force to counteract gravity's disturbance. The exact duty cycle was calculated from our initial current I_0 and the solenoid resistance.

To ensure a timely execution of these steps, we placed the controller code in a loop that waits for a short timed period before executing. Rather than using a delay statement, this technique accounts for the execution time of the control code and mandates that each loop has the same duration.

A complete version of the code can be found in Appendix A.

5 Performance

The system levitates successfully. If carefully situated, the magnets can float for up to 50 seconds.

Qualitative: In every tested trial, the permanent magnets spin quickly about the yaw axis. For this reason, we ensured that the magnet stack was as symmetric as possible. We also found that the system can correct for some small disturbance both vertically and laterally. Often, vertical disturbances will be "corrected" into more lateral motion. To mitigate lateral forces, we replaced some of the bottom permanent magnets with a steel nut.

Quantitative: Because of the system's issue with speed, it was not possible to plot or export data as the MagLev was operating. Thus collecting quantitative data was quite challenging. In the future, we would record videos of the operating MagLev with calipers inframe to measure the oscillation height and a range of viable starting positions. We would also attempt to measure the settling time after a disturbance, establishing a vertical and horizontal range of a corrected position, as well as the force of the disturbance itself. In sum, this data could have yielded interesting quantitative results about the capacities of our MagLev.

6 Process Considerations

We began this project by implementing a bang-bang controller. This allowed us to gain an intuition for an appropriate initial height and how many magnets to use. We also realized that the force of repulsion was too strong for a stable system, and pivoted to alternating between on and off signals only.

Then we attempted another form of a PD controller. We thought that we should use the sensor reading error to calculate what incremental change is needed in the voltage, and alter the power of the solenoid accordingly. This controller did not work, and we believed at the time that this was due to amplifying the Hall sensor voltages to match the voltage range of the solenoid. However, we pivoted from this technique before implementing the moving average filter, which may very well have improved its performance.

We also considered subtracting the effect of the solenoid from the Hall voltage in order to control for our range of PWM inputs. We compared sensor readings on a range of distances and solenoid duty cycles (as in Figure 5) but found that this difference was negligible.

7 Areas of Improvement

If we had more time and foresight, we would have rigorously tested our model's response to various inputs. By measuring the system's step response, we could have verified the order of the system as well as its dominating poles, further validating our model.

We realized late into the project that our amplifier was broken and was not affecting our Hall sensor readings. If we had amplified this signal, we could have gotten smoother derivatives which would have helped with the stability of the system and its response to disturbances.

Lastly, and most importantly, we should have recorded the data we collected throughout this experiment for more quantitative results. This would reduce the amount of time spent tuning the controller.

8 Conclusion

In the end, we implemented a proportional-derivative controller that partially stabilized the system. Our magnet could withstand small vertical disturbances and would remain stable for a significant amount of time.

Team Member Contributions: 50/50

A Appendix

```
#define target            390        // Target hall effect sensor reading
#define filterFactor      5          // Number of samples in running average
#define dT                1          // PWM update interval, in milliseconds
#define hallPin           1          // Analog pin for Hall Effect sensor
#define in2               5

#define minPWM            0           // Minimum duty cycle (0V)
#define maxPWM            255         // Maximum duty cycle (12V)
#define baselinePWM       128         // Baseline duty cycle for low setting -
    currently rounded midpoint of min and max PWM (6V)

int currentPWM = 0;                   // Current duty cycle to drive solenoid
int lastError = 0;                    // Last calculated error for derivative term
int lastSensorValue = 0;              // Last calculated sensor value for running
    average
signed int nextPIDCycle = 0;          // For smooth response, fix the speed that
    the control loop operates.

int k = 45.5;
float kp = 30.05*k;
float kd = 1*k;

int roundValue(float value) {
  return (int)(value + 0.5);
}

void setup(){
    pinMode(hallPin, INPUT);
    Serial.begin(9600);
}


void loop() {
    // Writing to the solenoid should happen at consistent time step. Here we
       compare the current time to PID cycle wait time. If negative, the time
       hasn't passed yet!
    if(0 <= ((typeof(gNextPIDCycle))millis() - gNextPIDCycle))
    {
      // Update time for next PID cycle
      nextPIDCycle = millis() + dT;

      // Update running average with current sensor reading
      lastSensorValue = roundValue(((lastSensorValue * (filterFactor - 1)) +
          analogRead(hallPin)) / filterFactor);

      // Difference between current and expected values (for proportional term)
      int error = target - lastSensorValue;

      // Slope of the input over time (for derivative term). This is called
         Derivative on Measurement, as opposed to the more normal Derivative on
         Error. Used to reduce "derivative kick" when changing the set point, not
          a huge deal at our frequency
      int dError = error - lastError;
```

```
45
46        // Subtract output of PD controller from baseline to get PWM output
47        currentPWM = baselinePWM - roundValue((kp*error) + (kd*dError));
48
49        // Since analogWrite wraps values out of range, we need to constrain our
              PWM between max and min
50        currentPWM = constrain(currentPWM, minPWM, maxPWM);
51
52        // Write PWM to solenoid
53        analogWrite(in2, currentPWM);
54
55        //Store for next calculation of dError
56        lastError = error;
57      }
58
59      //Wait for next PID update cycle... read the sensor value for filtering
60      else {
61        //Weighted average function takes previous samples, replaces one with the
              current sensor value, and averages over the total
62        lastSensorValue = roundValue(((lastSensorValue * (filterFactor - 1)) +
              analogRead(hallPin)) / filterFactor);
63      }
64  }
```