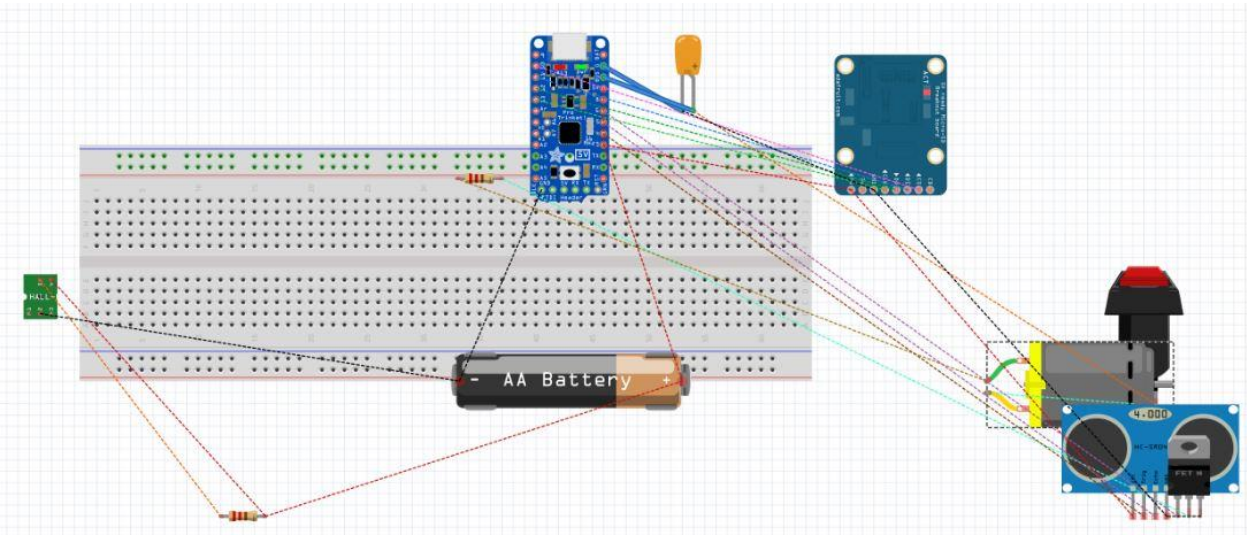
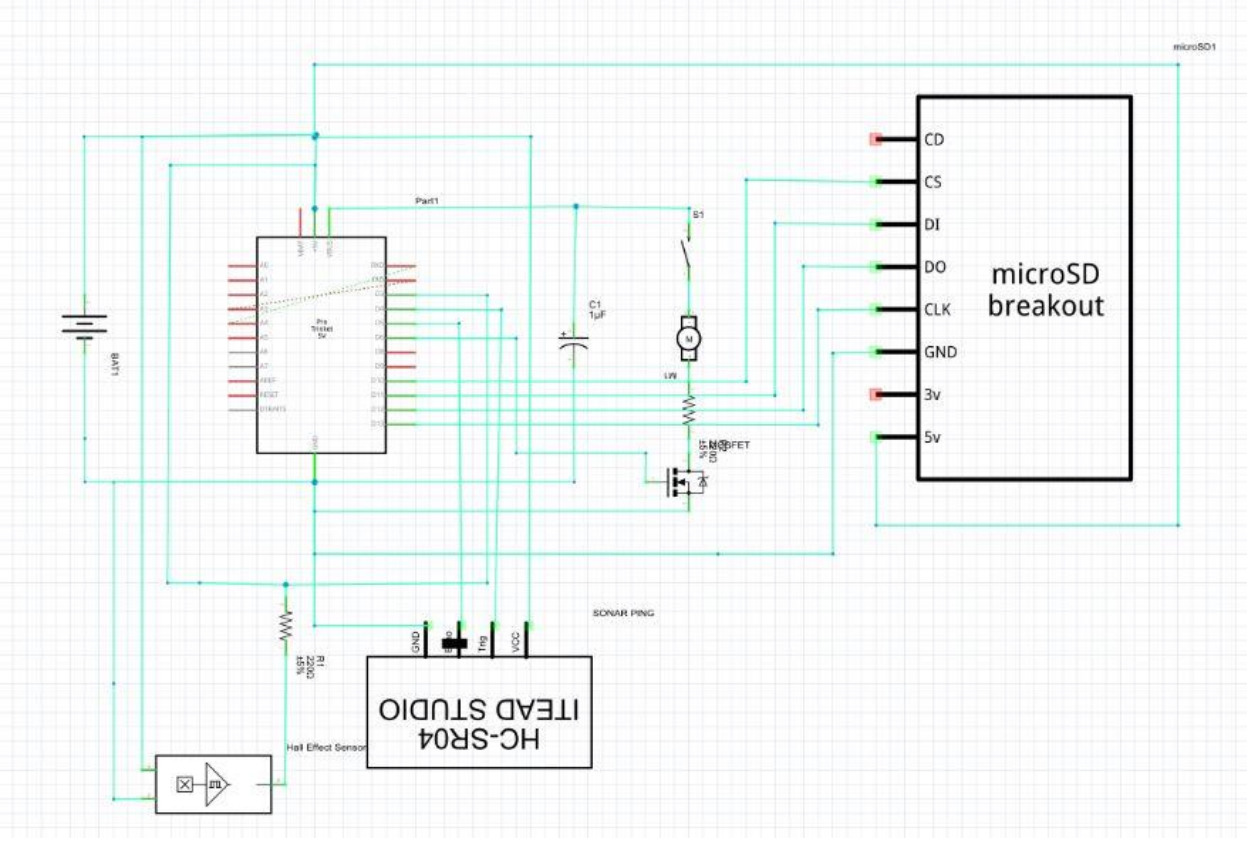
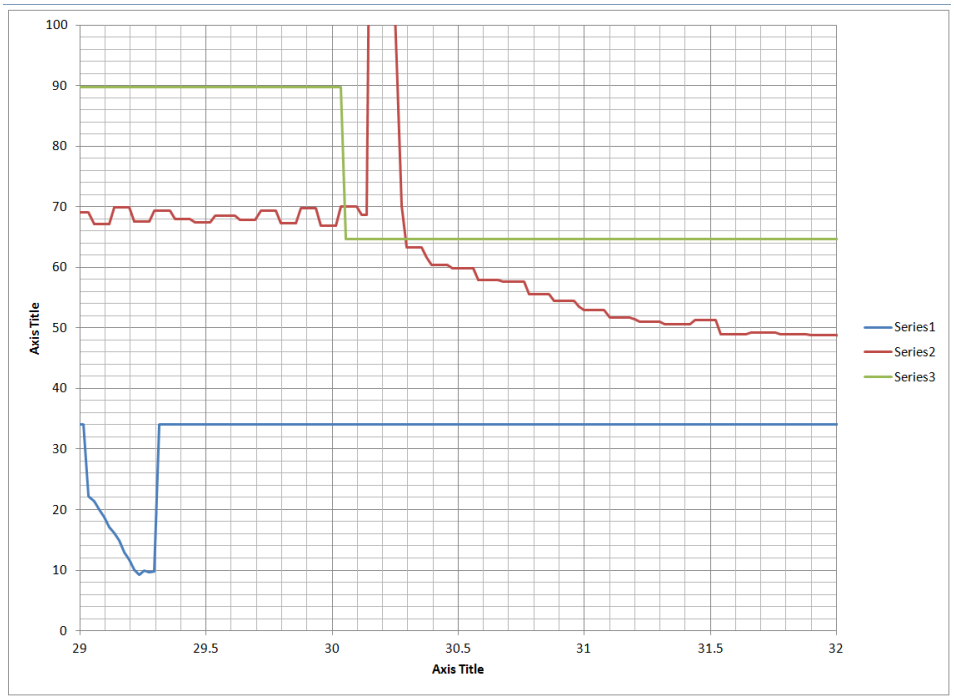
The first step in this project is to understand our train circuitry. Here's the schematic for the entire train system:



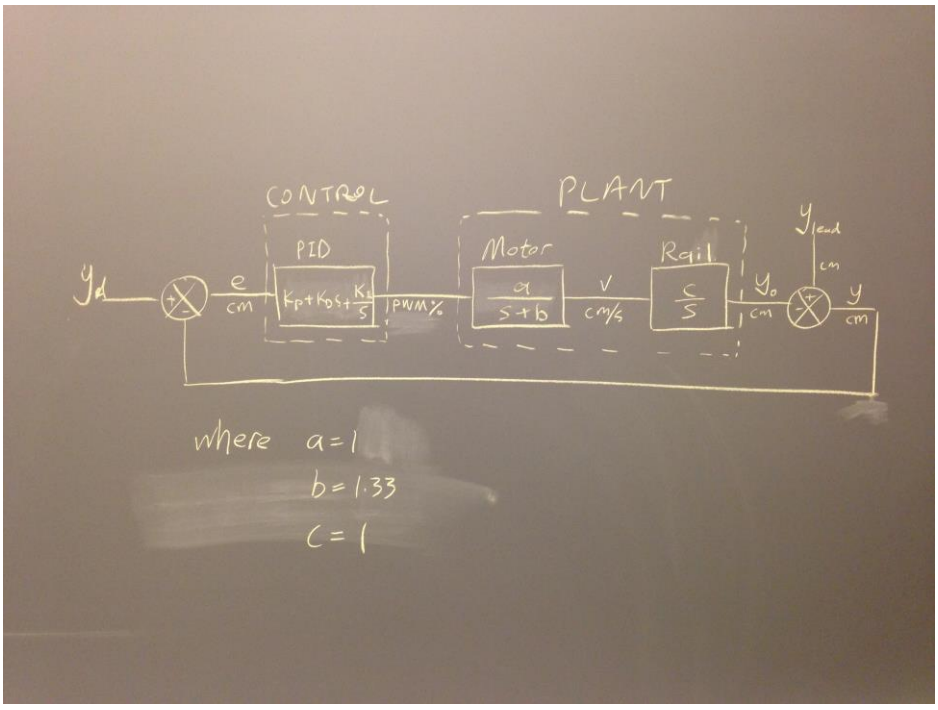


The second step in this project is to find a good model for the train dynamic response, specifically a transfer function from motor PWM input to the actual train speed. We did a step test on the train motor and gathered speed data from the speed sensor. The following graph shows the step response of the train:

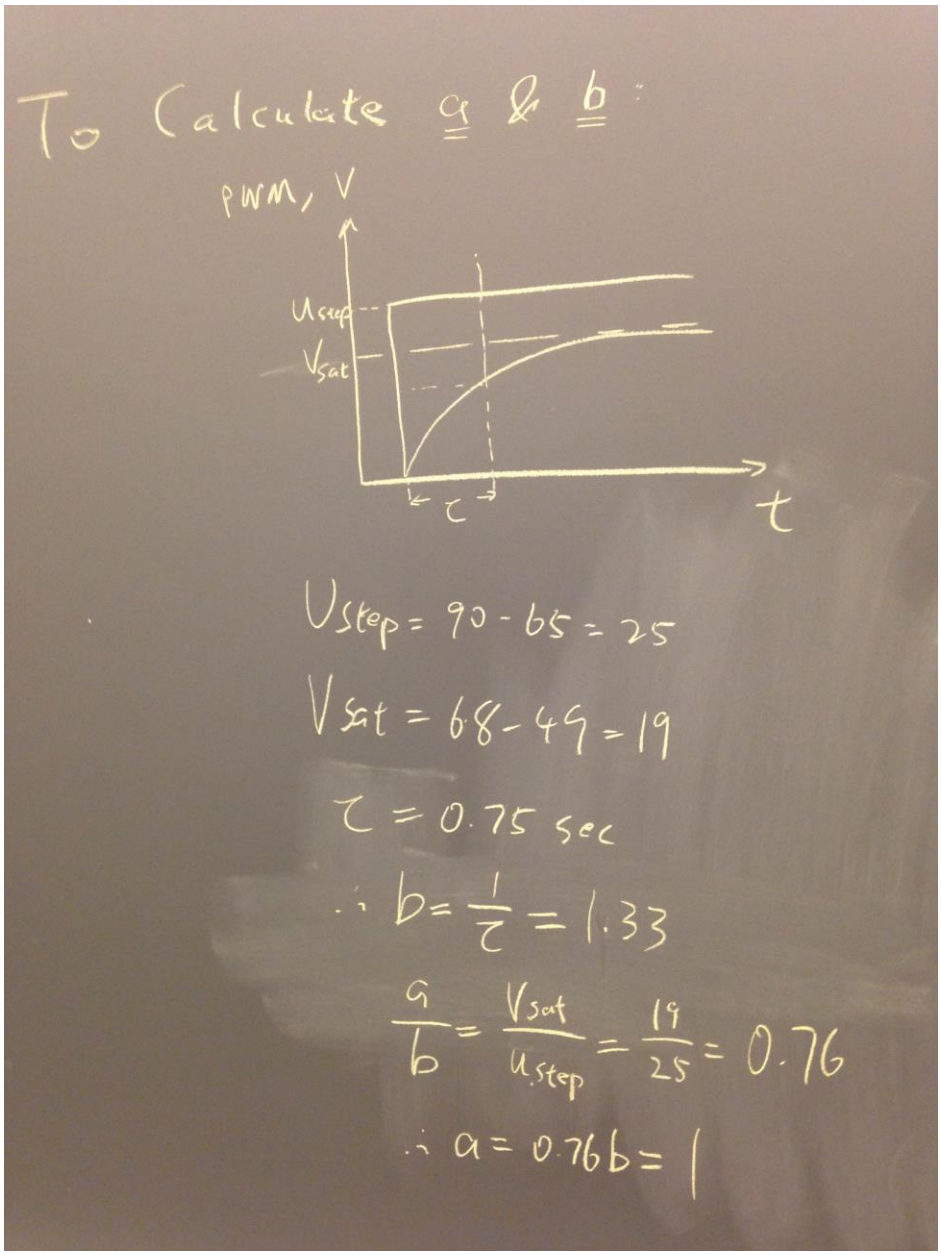


The green line is the step input, specifically the motor PWM percentage changed from 90% to 65%. The red curve is the step response, specifically the motor speed in cm/sec. We ignore the spike right after the step input because it is probably a misreading from the speed sensor. The step response looks like a first order system with a transfer function that looks like a/(s+b). We will just assume that this is the train model. Now we will try to calculate the parameters and get the model for the entire system.

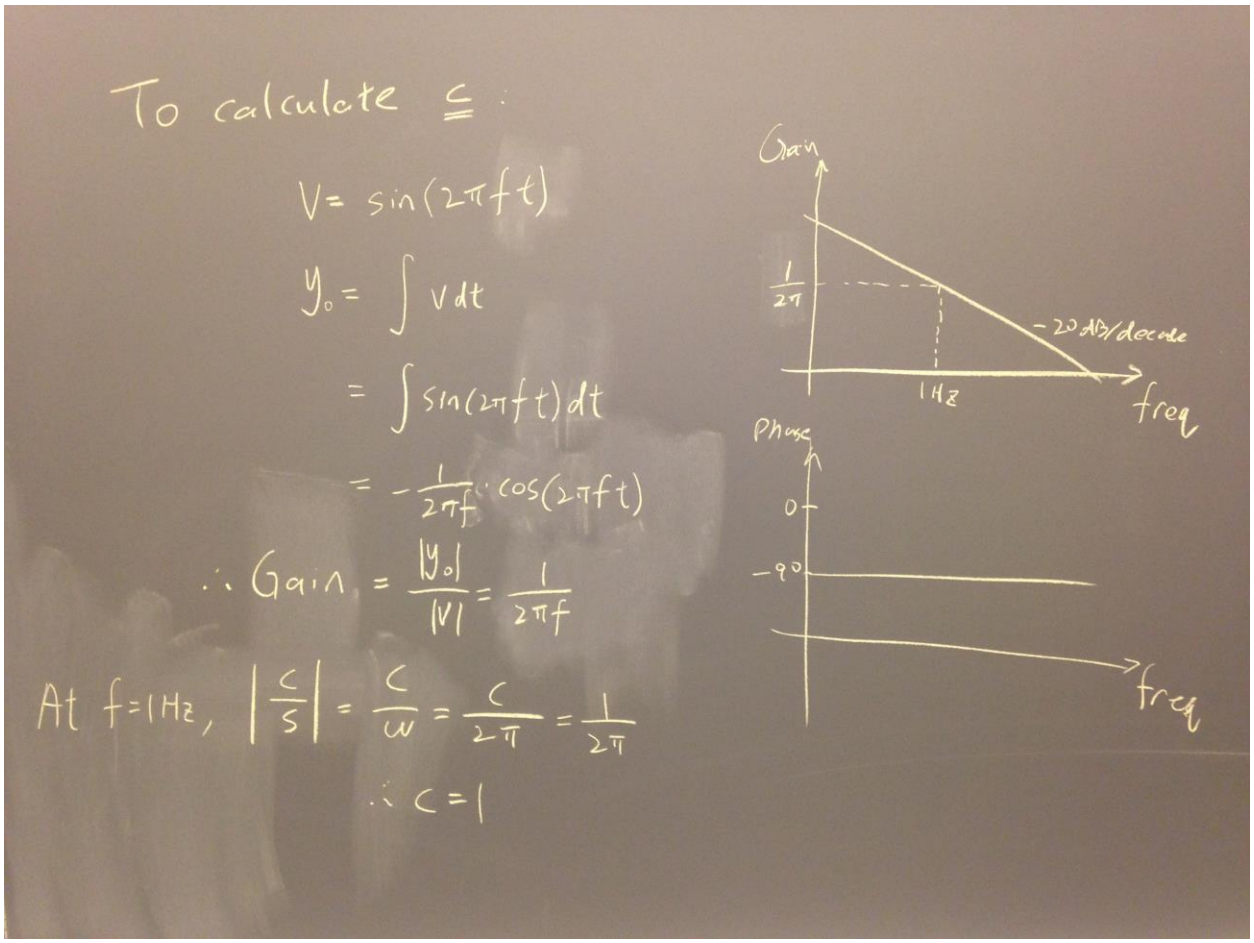
The system block diagram is as follows:



The rail block is just an integrator that integrates the speed of the train and get the distance of the train. Y_d is the desired gap with the lead train. Y_{lead} is the lead train distance. We choose the use a classic PID controller. After setting this model up we need to calculate parameters *a*, *b*, and *c*.



From the excel graph above we can roughly estimate *a* and *b* as above.

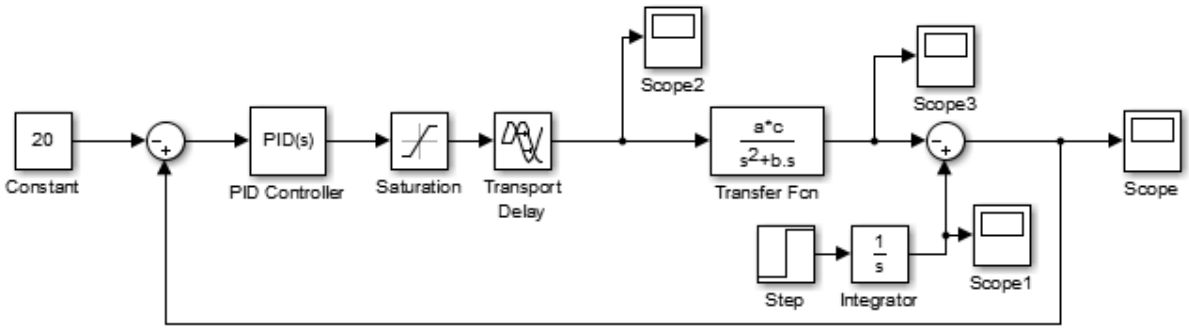


The rail block is just an integrator from speed (cm/s) to distance (cm). This is a simple math deduction as above to determine parameter c.

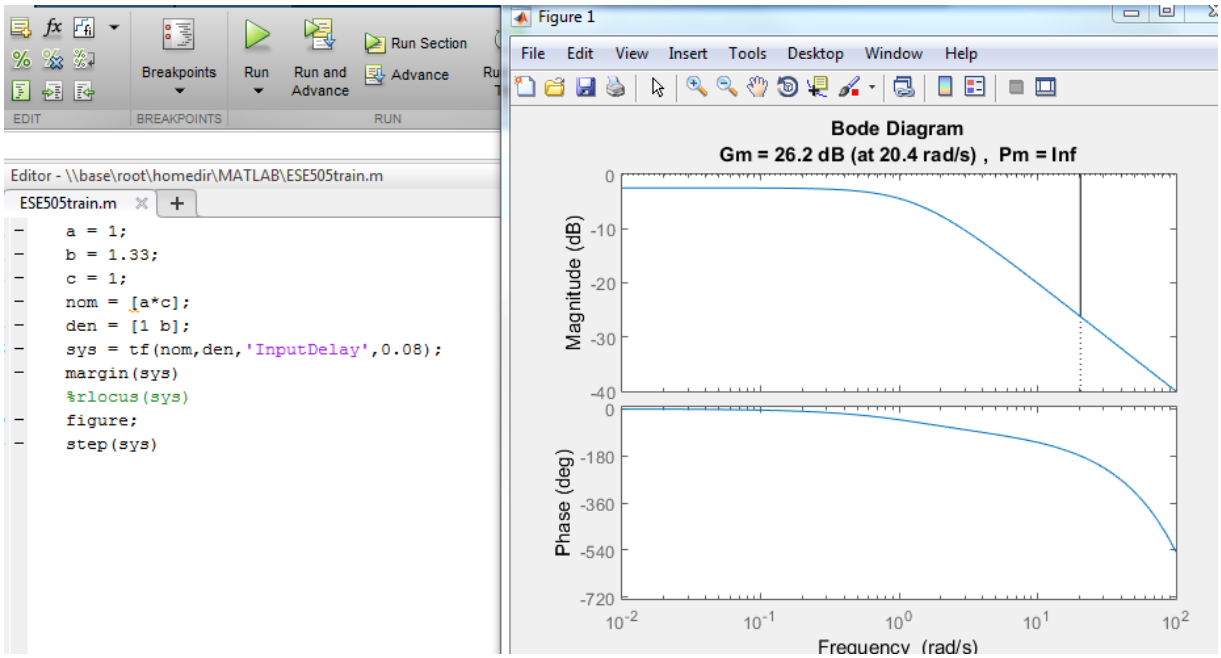
In the end we found that a = c = 1, b = 1.33.

With this model set up, we can build it in Simulink and run some simulations.

Simulink Model:



Here we set the constant gap to 20 instead of 10. Since the plant model is a second order system with time delay at around 80 millisecond, we can plot the bode plot in MATLAB and calculate the ultimate gain and the period of the system at neutral stability.

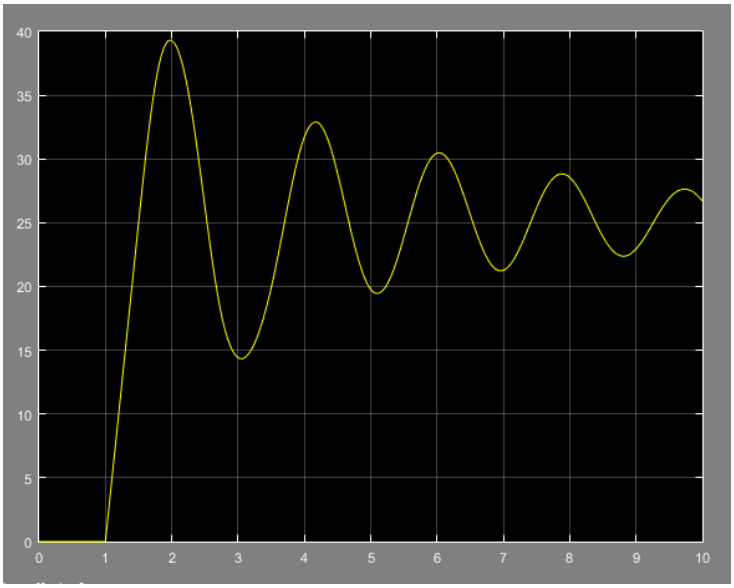


We calculated $K_u = 20$ and $T_u = 0.31$.

We tune the PID controller based on these two factors using the Ziegler-Nichols method. Specifically we set $K_p = 0.6*K_u$, $K_d = 0.5*T_u$ and $K_i = 0.125*T_u$.

Here are some simulations we did in Simulink. We set up an input signal to simulate the speed of the lead train and send it to the entire system to see its response to it.

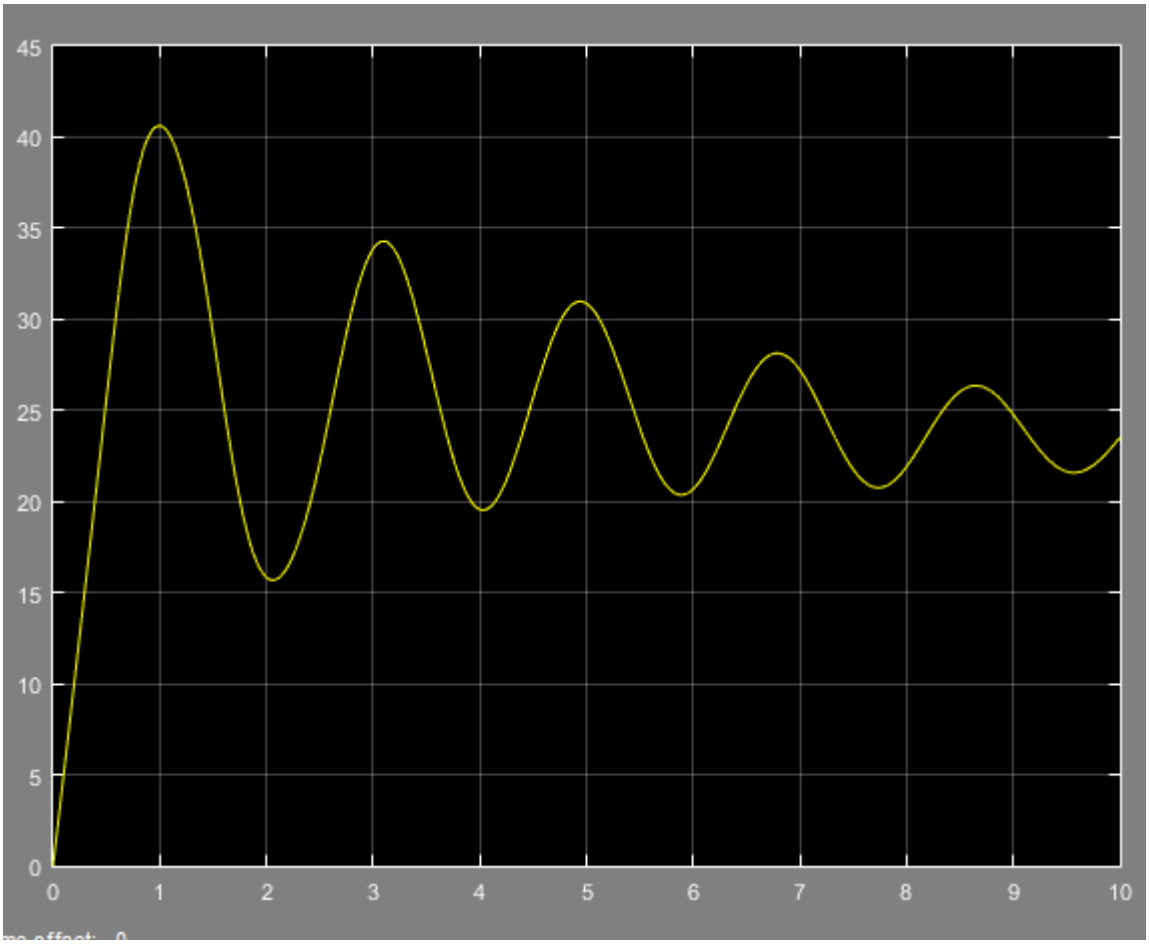
Step input:



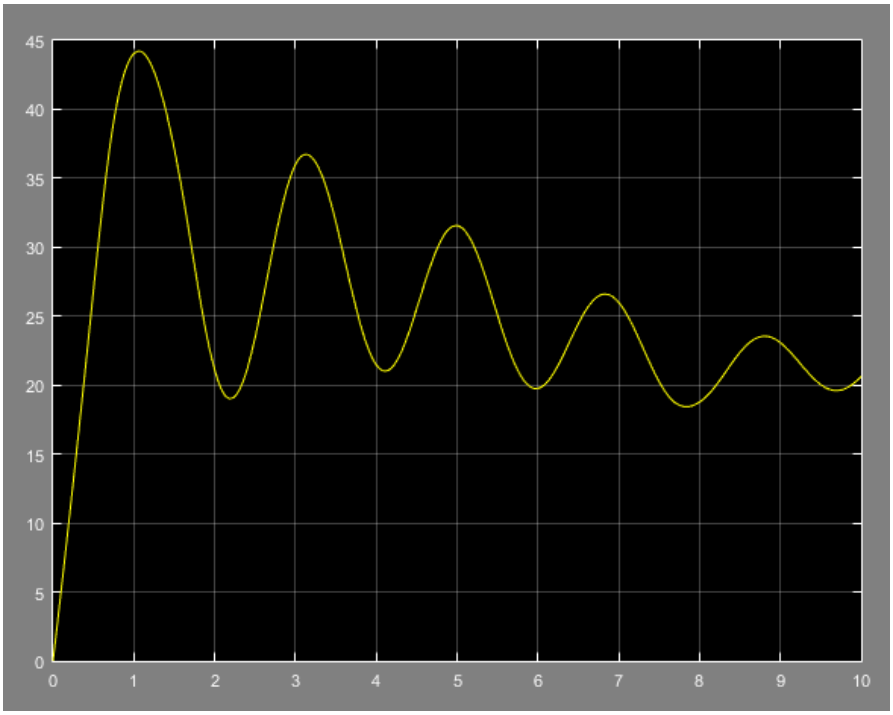
Note that the system converges to around 20 after the step input of the lead train. The system is stable.

Now we set the input signal to sine waves. We test the sine waves at different frequencies and different amplitudes.

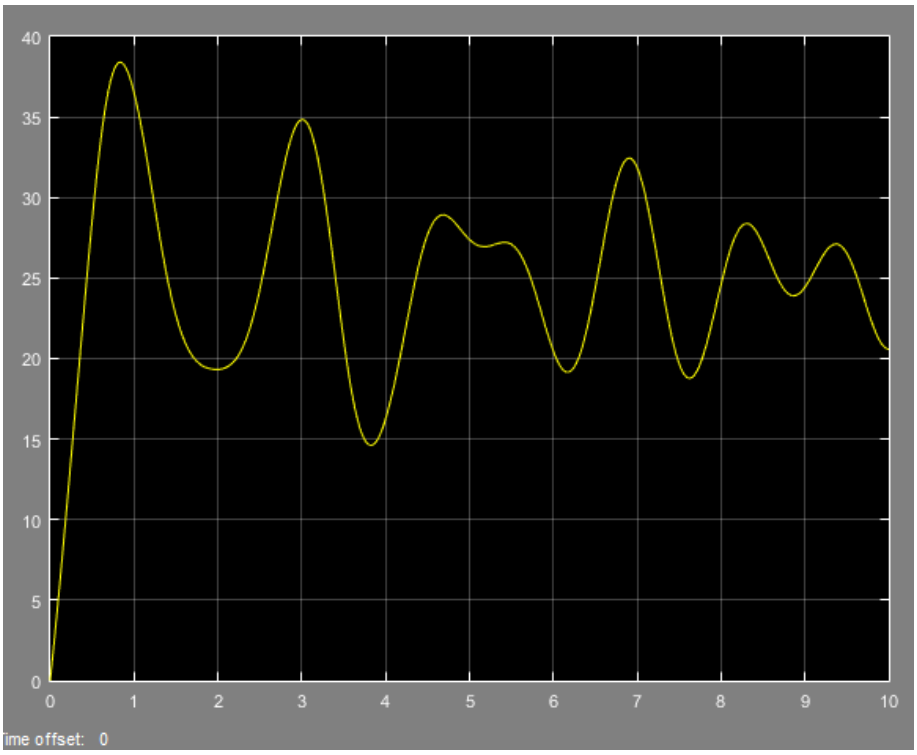
Frequency = 0.5Hz, Amplitude = 10cm/s, Velocity Offset = 50cm/s.



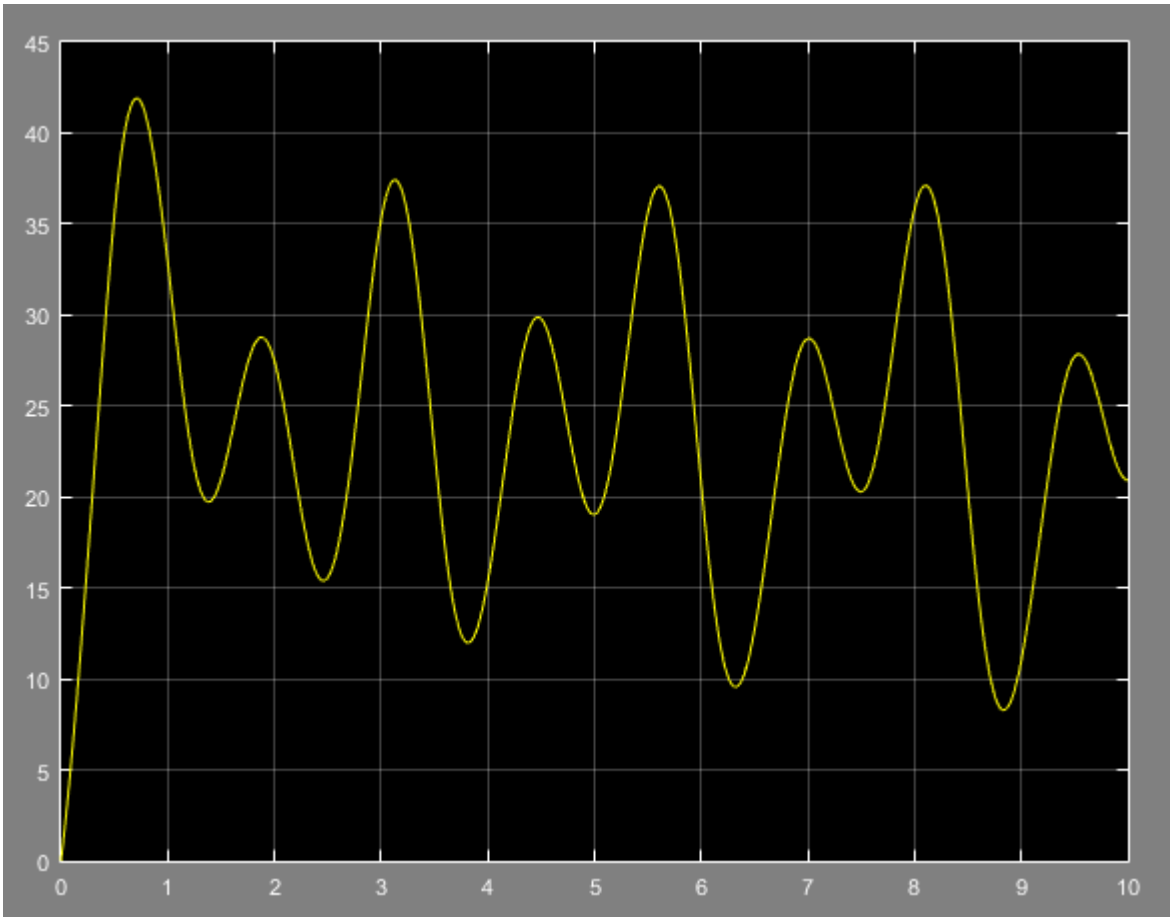
Frequency = 0.5Hz, Amplitude = 30cm/s, Velocity Offset = 50cm/s.



Frequency = 5Hz, Amplitude = 10cm/s, Velocity Offset = 50cm/s.



Frequency = 5Hz, Amplitude = 30cm/s, Velocity Offset = 50cm/s.



We see that at low frequencies, the output converges to 20 slowly and stably. At high frequencies, the input time delay kicks in and increases the non-linearity of the system. That's why we see multiple frequency component in the output signal where the input signal is composed of only one frequency component. The system is unstable at high frequency and high amplitudes. Note that the amplitude of the input sine wave does not affect the output as much as frequency.

The non-linearity of the system comes from the input time delay and the motor control signal range constraint. There is a limit of how fast the train can go. Also the motor control is not perfectly linear as we assumed it is. So in the Simulink model, we added the Saturation block and Transport Delay block to account for these non-linearities.

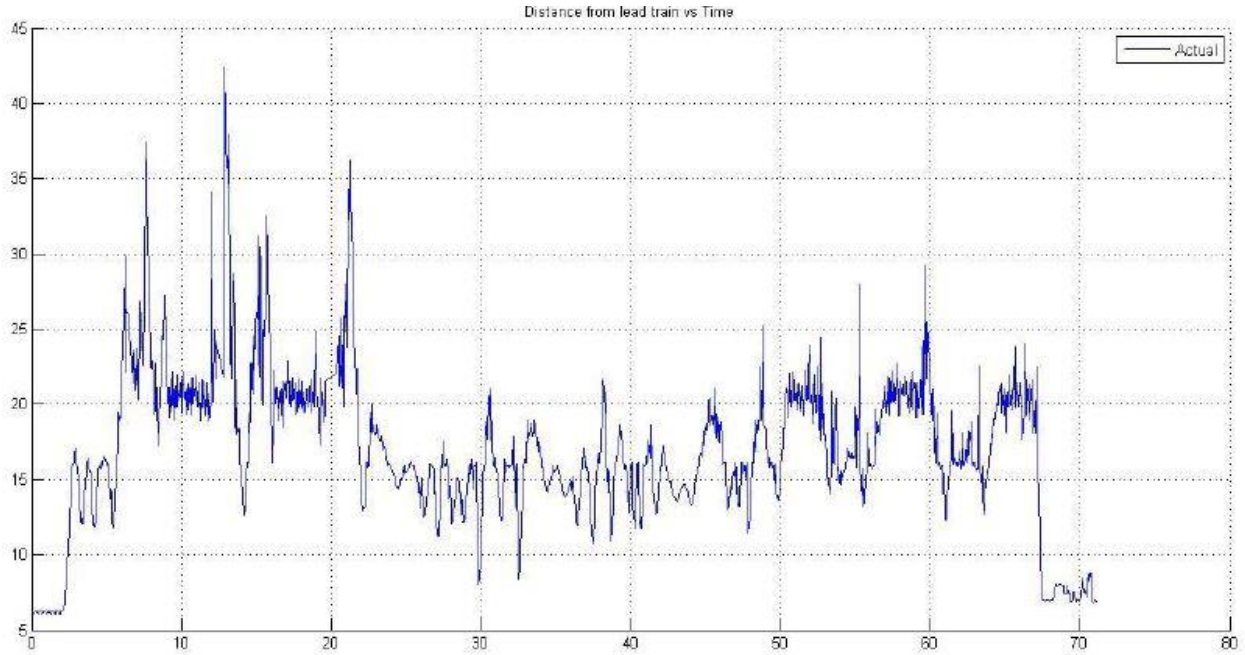
We use PID controller as the compensator and tuned it using the Ziegler-Nichols method. We implement the digital controller in Arduino by writing the following code:

```
byte train_control_law(double y, double v)
{
  double train_control_pct;
  byte train_control_pwm;
  float Ku, Kp, Ki, Kd, Tu, error, errorPrev;
  int yd=20;
  static float errorSum=0;
  static float erroPrev=0;
  Ku=20;
  Tu=0.31;


  Kp=0.6*Ku;
  Kd=Tu/2;
  Ki=Tu/8;
  error= y - yd;//14
  train_control_pct = 40+(Kp*error) + (Kd*(error-errorPrev)/deltaT_ms) + (Ki*errorSum) ;
  //train_control_pct = 70;
  errorPrev= error;
  errorSum= errorSum + error;


  train_control_pwm =constrain(2.55 * train_control_pct,0,255);
  return train_control_pwm;
}
```

The data recorded on the SD card is obtained as follows:

```
Train Control Data Log
210,4.11,26.72,153
230,4.27,26.72,153
250,4.27,26.72,153
270,4.27,26.72,153
290,4.27,26.72,153
310,4.27,26.72,153
330,4.27,26.72,153
350,4.27,26.72,153
370,4.27,26.72,153
390,4.68,26.72,153
410,4.68,26.72,153
430,4.68,25.70,153
450,4.68,23.51,153
470,4.68,21.75,153
490,4.68,20.16,153
510,4.68,18.78,153
530,4.68,17.64,153
550,4.68,16.58,153
570,4.68,15.69,153
590,4.68,14.84,153
610,4.68,14.12,153
630,4.68,13.43,153
650,4.68,12.84,153
670,4.68,12.27,153
690,4.68,11.77,153
710,4.68,11.29,153
730,4.68,10.87,153
```

On plotting the data of the distance of the follower train from the lead train, the graph is obtained:



The distance keeps fluctuating around the same point. The video links below show the response of the train to an obstruction placed in front. The code is implemented on two trains and the video of the same is shown below.

1) https://drive.google.com/file/d/0B1ydgUsPuikuVzdsVDkyNkdXQVU/view?usp=sharing

2) https://drive.google.com/file/d/0B1ydgUsPuikua2c0aXZUbXlISVU/view?usp=sharing